December 2020

Reasoning About User Feedback Under Identity Uncertainty in Knowledge Base Construction

Ariel Kobren

REASONING ABOUT USER FEEDBACK UNDER IDENTITY UNCERTAINTY IN KNOWLEDGE BASE CONSTRUCTION

A Dissertation Presented

by

ARI KOBREN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2020

College of Information and Computer Sciences

REASONING ABOUT USER FEEDBACK UNDER IDENTITY UNCERTAINTY IN KNOWLEDGE BASE CONSTRUCTION

A Dissertation Presented

by

ARI KOBREN

Approved as to style and content by:

_____

Andrew McCallum, Chair

_____

Akshay Krishnamurthy, Member

_____

Alexandra Meliou, Member

_____

Barna Saha, Member

_____

Christopher Ré, Member

_____

James Allan, Chair of the Faculty
College of Information and Computer Sciences

DEDICATION

To Mama and Abba.

ACKNOWLEDGMENTS

בן זומא אומר: איזהו חכם , הלומד מכל אדם.

Ben Zoma said: who is wise? A person who learns from everyone.

Ethics of our Ancestors 4:1

For me, the pursuit of a doctorate in computer science has been, in large part, about learning. Starting with the most obvious: my research is centered on machine *learning*. At the start of my Ph.D., I wanted to create new machine learning knowledge that others could learn from—in a way, to be a teacher. I quickly learned that making such a contribution required that I first become a student of classic computer science research as well as the bleeding edge of innovation in artificial intelligence. But these technical topics were only part of what I needed to learn in order to carry out my dissertation research. As I have come to understand, the life lessons I learned from my teachers, colleagues, students, family and friends were, perhaps, even more critical.

Throughout my Ph.D., my advisor, Andrew, has taught me many important lessons; one of the most important is about *ambition*. Especially during my first few years as a student in IESL (Andrew's laboratory), I was regularly surprised by the large number of challenging goals Andrew set for his students. I remember often thinking that accomplishing (all) those goals would be impossible. But, in many cases I was wrong. As I learned from Andrew, setting one's sights on seemingly unreachable goals is the only way to discover the extent of what can be accomplished. Moreover, I learned that accomplishing lofty goals always begins the same way: trying. Andrew: thank you for being my role model and my friend, and for the immense, positive impact you have had on my life.

I have also learned a tremendous amount from my collaborators. My closest collaborator—Nick Monath—has taught me what *work hard* really means. I continue to be impressed by Nick's work ethic and ability to juggle what seems like an uncountably infinite number of simultaneous research projects, service efforts, and mentees. Thank you to Akshay Krishnamurthy, who taught me the value of deliberately trying to *break* a research idea, i.e., find its flaws. Initially, having Akshay break all of my ideas was disheartening; but, these exercises lead to some of my best work. I have learned a great deal from the current and former members of my lab, IESL; for example: how to do collaborative research, how to be generous with one's research hours and effort, and about countless technical topics. Thank you to the students I have mentored: you have taught me invaluable lessons about leading successful research projects. And, thank you to my thesis committee: Andrew, Akshay, Alexandra, Barna and Chris. You have provided me with many interesting suggestions for improving my dissertation work and have even helped me to improve unrelated research projects of mine.

I've also learned valuable life lessons from my friends and the communities of which I have been a part. Especially in the first few years of my Ph.D., Noah Slovin taught me some of life's most important lessons: the thrill of night-biking and night-hiking, the joy of a fried egg with a runny yolk, that it takes much longer for food to go bad than I initially thought (and that you should never throw out food before smelling—and ideally, tasting—it), various facts about ArgGIS and fluvial geomorphology, and, generally, ingenuity. Thank you to Congregation Beth Israel in Northampton, where I learned the value of *Shabbat*—taking a day off from my research to work on myself and my relationships with others. Also, to Garrett Bernstein and Jonathan, Elizabeth and Elanor Dubinsky: thank you for teaching me hospitality. You all graciously provided me a place to sleep a few nights a week, month after month, when I lived outside of Amherst but returned for regular meetings. Finally, Don Darnell and Robbin Derry: thank you for teaching me about being an excellent neighbor. Baking Shilpa and me cookies, offering us an impromptu loaf of homemade bread right

out of the oven, lending us furniture when we hosted large gatherings, and offering to host our friends when we didn't have enough space for them to sleep are just a few of the ways you've strengthened our relationship and offered me support at times I really needed it, whether you knew it then or not.

My parents are my longest running and most impactful teachers. Abba: thank you for teaching me how to write. As a child practicing my writing with you, I *never* imagined how important this skill would be throughout my life; I cannot overstate its usefulness. The same two sayings you used to recite to me are still as true as ever: i) *good writing is rewriting what you've already rewritten*, and ii) *writing is evidence of thinking*. Mama: thank you for teaching me grit. Without it, I am certain I would not have withstood all the challenges I encountered during my doctoral research. In many aspects of your life, you are a shining example of the saying: *never give up*. To Shira, Jason, Gavi and Noam: thank you for being exceptionally supportive family members, for your understanding throughout my Ph.D., and for teaching me the wonders of New York City. To Amma and Nanna: thank you for teaching me to make Indian food (which has become one of my hobbies), but also for making Shilpa and me care packages, taking us on vacations and being *so* helpful with childcare.

Finally, thank you to my lifelong partner (and wife), Shilpa, and our son Tal. Tal (a.k.a., *boychick*): despite your current inability to communicate with me in Hebrew or English, you have taught me perspective. Even during especially trying times, hearing you laugh or seeing you smile has made *me* smile, without fail. You are my constant reminder not to forget about the most important things in life. And, perhaps most importantly, to Shilpa: I have learned countless, invaluable life lessons, either with you as my teacher or together as students—like when we met as pair programmers in Comp40. While the things I've learned with and from you are far too numerous to describe, I think they can be neatly summarized by saying that you have taught me love. You have shown me unwavering support, understanding, and friendship throughout our lives together. You are there for me, always, in good times and

tough times, and willing to encourage and even participate in any endeavor I wish to take on, be it sourdough bread-making, backpacking, canning enough pickles to last an entire year, or Ph.D. research. While this dissertation marks the end of one chapter of our adventure together (which could be called "The Doctorates"), I'm excited for what life has in store for us next, and all the things we will learn together.

ABSTRACT

REASONING ABOUT USER FEEDBACK UNDER IDENTITY UNCERTAINTY IN KNOWLEDGE BASE CONSTRUCTION

SEPTEMBER 2020

ARI KOBREN

B.Sc., TUFTS UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McCallum

Intelligent, automated systems that are intertwined with everyday life—such as Google Search and virtual assistants like Amazon's Alexa or Apple's Siri—are often powered in part by knowledge bases (KBs), i.e., structured data repositories of entities, their attributes, and the relationships among them. Despite a wealth of research focused on automated KB construction methods, KBs are inevitably imperfect, with errors stemming from various points in the construction pipeline. Making matters more challenging, new data is created daily and must be integrated with existing KBs so that they remain up-to-date. As the primary consumers of KBs, human users have tremendous potential to aid in KB construction by contributing feedback that identifies spurious and missing entity attributes and relations. However, correctly integrating user feedback with an existing KB is complicated by the necessity to resolve *identity uncertainty*, i.e., uncertainty regarding to which real-world entity a piece of data refers. Identity uncertainty abounds in the collection of raw evidence

from which a KB is built. Moreover, it also gives rise to identity uncertainty *in user feedback*, when KB entities, which were affected by user feedback, are split or merged.

In this dissertation, we present a continuous reasoning framework capable of integrating user feedback with a KB, under identity certainty. To begin, we introduce GRINCH, an online entity resolution (ER) algorithm—with provable correctness guarantees—capable of merging and splitting KB entities as new data arrives. We show that GRINCH is efficient and achieves state-of-the-art performance in ER as well as in clustering. Next, we propose a method for using GRINCH to resolve identity uncertainty in a KB's underlying data as well as in user feedback. Our approach is based on representing user feedback as *mentions*, i.e., first class KB objects that participate in all parts of KB construction. Furthermore, we introduce a structured representation for feedback comprised of *packaging* and *payload*, which facilitates recovery from KB errors that stem from both identity uncertainty and noisy data. Finally, we evaluate our framework's efficacy using data from the KB that supports `OpenReview.net`—a deployed, conference management system that solicits feedback from users. The demands of `OpenReview.net` lead us to develop XGRINCH-SHALLOW (XGS), a variant of GRINCH that builds trees with arbitrary branching factors, and subsequently instantiates 60% fewer internal nodes than GRINCH. Empirically, we show that XGS is efficient, and is able to effectively utilize user feedback to improve the correctness and completeness of the `OpenReview.net` KB. We conclude with 7 concrete suggestions for future research on this topic.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Overview

Structured data repositories of entities and relations, known as knowledge bases (KBs), are increasingly prevalent as stand-alone resources and as central components of popular automated tools. For example, Wikidata [109], DBPedia [64] and YAGO2 [46] are all publicly available KBs of general knowledge, and both Google Search and Maps are reliant on the Google Knowledge Graph. Domain-specific KBs are also ubiquitous and even play a central role in some research areas, such as digital libraries [110].

Whether they are constructed automatically or by hand, KBs are often incomplete and noisy. For example, it has been reported that 71% of people in Freebase are missing a place of birth attribute and 75% have no known nationality [26]. Similarly, while YAGO2 is estimated to be 95% accurate, this translates to roughly 22 million incorrect facts involving 9.8 million entities [46]. The vast research in cleaning and correction of databases is further evidence of the permeation of errors throughout the construction of KBs in multiple domains [26, 67, 115, 27].

As the primary consumers of KBs, human users have significant potential to aid in KB construction and maintenance. From a user's standpoint, a KB contains a set of entities, each entity possessing attributes and optionally participating in relationships with other entities. Errors appear to users as spurious and missing attributes and relationships. However, the data that gives rise to a KB is a collection of raw evidence, which can be understood as *mentions* that require clustering by entity resolution (ER) into a set of *inferred entities*. The attributes and relations of the inferred KB entities with which the user interacts are

drawn from this underlying clustering of the mentions. Therefore, the spurious and missing attributes and relationships, may stem from a variety of sources, including: noisy mentions produced by information extraction, mistakes in ER, missing data, etc.

In light of new data that is continually being added to a KB, inferred KB entities may change. Specifically, the arrival of new mentions and user feedback can trigger modifications of the underlying mention clustering, resulting in the creation of new inferred entities, removal of previously inferred entities or alteration of the existing inferred entities' attributes and relations. The volatility of the underlying mention clustering poses a formidable challenge to the task of integrating user feedback with KB content, especially when the precise targets of feedback are unknown, a phenomenon known as *identity uncertainty* [77, 84]. We provide the following example to ground our discussion of user feedback that exhibits identity uncertainty.

Example 1. *[Feedback Exhibiting Identity Uncertainty] Consider a KB of scientists presenting the scientist's topical expertise, affiliation history, scientific collaborations and previous work—all gathered from research papers and other mentions of the scientists. While interacting with the KB, a user notices the entity* `Fernando Pereira` *and submits feedback asserting that* `Fernando Pereira` collaborated with a scientist named John Blitzer. *Later, another user notices an error in* `Fernando Pereira`*'s affiliation history, which stems from* ER *incorrectly merging mentions of a* second *Fernando Pereira with the mentions of the existing* `Fernando Pereira` *KB entity. In an attempt to correct the error, which manifests as an incorrect affiliation, the user submits feedback claiming that* `Fernando Pereira`*,* who is affiliated with the organization Google, was never affiliated with the Instituto de Telecomunicações in Portugal. *This second piece of feedback signals to the KB that the mentions comprising* `Fernando Pereira` *belong to at least 2 distinct, real-world entities. The KB proceeds to split the corresponding cluster of mentions into two inferred entities. But, after the split, the decision of to which of the clusters the* first *piece of*

feedback applies is not entirely clear because of uncertainty with respect to the true identity of the Fernando Pereira mentioned in that feedback.

In this thesis, we present a framework for integrating user feedback amidst identity uncertainty throughout KB construction. Our framework is comprised of 3 key building blocks. The first is a new, online, hierarchical clustering algorithm for ER (Chapter 3) that supports deterministic, non-local rearrangements of the hierarchy in response to newly added data. This feature is crucial for recovering from ER errors, which can require splitting and merging of previously inferred entities (as in the example above), and distinguishes our work from previous approaches. The second building block is the notion that each piece of user feedback should be represented *as a mention* and participate as a first-class object during ER (Chapter 4). Under this approach, when an inferred entity that was previously the target of user feedback is split–as in the example above–the decision about the new target of the feedback is naturally handled by our new ER algorithm. The final component in our framework is the structuring of user feedback with two sets of attributes—*packaging* and *payload*—that facilitate recovery from KB errors stemming from different sources, and which require distinct styles of correction (Chapter 4). We present empirical results on synthetic and real-world data KBs that include user feedback as well as theoretical results with respect to our framework (Section 5.4). As this thesis helps lay a foundation for the study of methods for integrating user feedback with KBs under identity uncertainty, we conclude with a discussion of various paths for future investigation.

1.2 Thesis Statement and Summary of Contributions

In this thesis, we argue that user feedback is a key ingredient in maintaining growing KBs. Moreover, if identity uncertainty abounds among the raw data used to build a KB, effective integration schemes must consider identity uncertainty of user feedback. Furthermore, it is vital to consider the relationship among the possible KB error types, the breadth of corresponding valid user feedback, and the method used for feedback integration in order to

design a representation of user feedback that leads to effective and predictable integration. Formally,

Thesis Statement: *Effective utilization of user feedback during KB construction can be achieved by treating the feedback as raw evidence that may exhibit identity uncertainty, and choosing representations of the feedback that are designed to facilitate recovery from a wide range of possible KB error types.*

The primary contributions of this thesis are summarized below:

1. Design of a new, incremental, hierarchical clustering algorithm for ER that leverages both local and global rearrangements and an empirical demonstration that the algorithm performs well in practice (Chapter 3).

2. Development of a new notion of *separability* for clustering that is defined with respect to *any* model, and a theoretical guarantee that our incremental clustering algorithm constructs trees that contains the optimal clustering when the separability condition is met (Section 3.2).

3. *Attribute feedback*, and a proposal to represent user feedback as mentions that may contain attributes with *negative* weight (Chapter 4).

4. An empirical investigation on synthetically generated data showing that representing user feedback as mentions and allowing the integration algorithm to reason about identity uncertainty is superior to strategies that allow feedback to overwrite inferred entity attributes (Chapter 4).

5. *Packaging* and *payload* feedback representation that allow for recovery from ER errors as well as data corruption errors (Chapter 4).

6. Development of XGRINCH-SHALLOW, which extends GRINCH by allowing for arbitrary branching factor and also reduces the amount of required memory (Chapter 5).

7. Experiments that showcase the utility of our framework (XGRINCH-SHALLOW, edits as mentions, attribute feedback, and packaging and payload edit representation) with real user feedback collected via OpenReview.net (Section 5.4).

8. Identification of 7 concrete avenues of future work related to integration of user feedback during KB construction (Chapter 6).

1.3 Thesis Outline

This chapter (Chapter 1) serves as an overview of our primary focus: we describe our study of user feedback and its interplay with knowledge base construction, the problems that naturally transpire, and our approaches to these problems. In the next chapter, we survey the components of knowledge base construction pipelines, which sets the foundation for our technical contributions. We pay special attention to identity uncertainty in knowledge base construction, as well as its ramifications for user feedback. Chapter 3 details our first technical contribution, GRINCH: an incremental clustering algorithm that we use for entity resolution. We show positive experimental results on benchmark entity resolution datasets, as well as in the more general setting of clustering. Next, in Chapter 4, we dive into issues related to user feedback. We introduce attribute feedback, and provide a framework for integrating this style of user feedback under identity uncertainty with GRINCH. Our proposal is based on a representation of user feedback as raw evidence that contain a *packaging* and a *payload*. Finally, we study a real-world knowledge base that receives user feedback: OpenReview.net. We introduce XGRINCH-SHALLOW (Chapter 5), a GRINCH variant designed to use less memory and run on OpenReview.net data. Experimentation with XGRINCH-SHALLOW and its viability as a feedback integration algorithm are described in Chapter 5.4. In Chapter 6, we conclude with a summary of results and contributions, as well as avenues for future work.

1.4 Declaration of Collaborations

The following is published work that was done jointly with the named researchers:

- The preliminary ideas related to the treatment of user edits as mentions with packaging and payload were developed with Michael L. Wick and Andrew McCallum [118].
- Entity-centric attribute feedback initially appears in work with Nicholas Monath and Andrew McCallum [56].
- The GRINCH algorithm was designed jointly with Nicholas Monath, Akshay Krishnamurthy, Michael Glass and Andrew McCallum [79].
- Refinement of packaging and payload and the most recent experimental results on integrating user feedback under identity uncertainty is work done in collaboration with Nicholas Monath and Andrew McCallum [57].

CHAPTER 2

ROLES FOR USER FEEDBACK IN KNOWLEDGE BASE CONSTRUCTION

Knowledge bases (KBs) are structured data repositories, which are often used to support a plethora of services and technologies. As one example, the creators of one of the first major KBs—Cyc—argued that having access to a sizeable KB would help to endow artificially intelligent agents with common sense, thereby breaking the agents' brittleness and resolving "the AI bottleneck" [65]. However, the past few decades have shown that, while building useful KBs is possible, it is also challenging.

In this chapter, we describe knowledge base construction and outline the roles that user feedback might play in the process. We begin by discussing a broad family of techniques designed to facilitate the conversion of unstructured data into a structured, actionable representation. We pay special attention to *entity resolution* (ER), which poses one of the primary challenges of KB construction, especially when simultaneously reasoning about user feedback. Then, we provide a vision for how humans, who are significant consumers of KBs, can play a more active role in their construction, the related issues that arise.

To ground our discussions, we begin with a motivating example to which we will refer throughout the chapter.

Example 2 (A Knowledge Base of All Scientists)**.** *A primary inspiration for this thesis is the desire to build a knowledge base of* all scientists*. Such a KB would include living and deceased scientists, their grants and awards won, collaborations, areas of expertise, publications, biographical information and affiliation history. The KB of scientists would trace each of these elements for each scientist over time. Crucially, the KB must dynamically*

grow to include new information: emerging scientists, new grants and awards won, evolving collaboration networks, ebb and flow of expertise, the most recent publications and changes to affiliations.

The KB described in Example 2 is not only our goal. Examples of existing resources that partially address this challenge are many: ArnetMiner[1], ArXiv[2], Citeseer[3], DBLP[4], Google Scholar[5], OpenReview.net[6], ORCID[7], PubMed[8], Scopus[9], and the Web of Science[10]. Some of these resources are focused on particular domains (e.g., PubMed), others are focused on publications (e.g., ArXiv and DBLP), while others offer additional services not described in Example 2 (e.g., OpenReview.net). Yet, all are centered on cataloguing scientific knowledge and those who create it.

2.1 Knowledge Base Construction

A knowledge base (KB) is a structured database of *entities*, their *attributes* and the *relationships* among the KB entities. In Example 2, the *entities* include scientists, but also the papers they publish and the institution with which they are affiliated. The *attributes* of the scientist include things like: their date of birth, email addresses, homepages, etc.

[1]https://aminer.org/

[2]https://arxiv.org/

[3]http://csxstatic.ist.psu.edu/

[4]https://dblp.uni-trier.de/

[5]https://scholar.google.com/

[6]https://openreview.net/

[7]https://orcid.org/

[8]https://www.ncbi.nlm.nih.gov/pubmed/

[9]https://www.scopus.com/home.uri

[10]https://clarivate.com/products/web-of-science/

Relationships (among entities) include things like a collaboration between two scientists and the affiliation of a scientist with an institution.

Constructing such a comprehensive KB is a challenging task that has enjoyed significant research attention. While many varieties of KBs exist with a corresponding variety of construction methods, in the following subsections we broadly describe common sub-tasks of KB construction that are especially pertinent to our work. These sub-tasks include: information extraction, entity resolution, relation extraction, and canonicalization.

2.1.1 Information Extraction

Recall that KB construction refers to the process of converting unstructured and heterogeneously-structured information into a consistent, structured, actionable format. As such, the first step in KB construction is *information extraction*. The input to information extraction is a collection of unstructured (and heterogeneously-structured) data, which are often referred to as *documents*. The output of information extraction is a collection *mentions*, which are structured records that refer to particular KB entities.

In Example 2, a mention of a particular scientist could be a BibTeX record (i.e., a structured citation). Such a mention might include: the name of the target scientist, the names of some of the scientist's collaborators, and information related to one of that scientist's publications. In fact, a BibTeX record could be treated as a mention of *each* of the publication's coauthors, and of the publication itself. While some BibTeX (and similar, structured citation) records are readily available (e.g., those included in DBLP), others must be extracted from PDFs—a task known as *citation field extraction*. Citation field extraction has traditionally be performed using trained sequence models [5], but now, state of the art results are achieved by leveraging contextualized word embedding (specifically RoBERTa) [102, 72]. We specifically call attention to citation field extraction because, in Section 5.4, we perform experiments in the context of OpenReview.net, where many of the mentions are structured citations.

For completeness, we also touch on the more general task of extracting mentions from natural language text. For example, one well-studied sub-task is *named entity recognition* (NER), where the goal is to tag mentions of entities (i.e., people, places, organizations, etc.) in free text [38]. More concretely, in NER, the input is a sequence of tokens and the output is a sequence of tags, one for each token, where each tag indicates whether the corresponding token is the first token of in a token subsequence that refers to an entity, a non-beginning token in the subsequence, or not part of such a subsequence. There has been substantial research effort on this front, which we do not cover; the interested reader should see relevant surveys [81, 123]. A related sub-task is *entity typing*, where each token in the sequence is tagged with its type (e.g., person, location, organization, etc.). Research in entity typing has focused on methods that work with increasingly finer-grained types, and that allow each token to have multiple types [61, 70, 2, 25, 80, 98]. The results of NER and entity typing are mentions, their corresponding types, and context.

This thesis does not develop new methods for information extraction. Instead, we assume that the mentions are provided or readily available (i.e., they are available via a KB like DBLP). However, our contributions all rely on having an underlying set of mentions as input. As such, basic understanding of how mentions come to be and what types of information they contain is highly relevant and important.

2.1.2 Relation Extraction

In addition to detecting entity mentions and labeling them with the corresponding types, KBs store relationships among entities. These relationships can be discovered using relation extraction techniques. In a traditional approach, a multi-class classifier is trained to predict what relationship (if any) is expressed by a sentence containing two named entities [50]. Semi-supervised [14], distantly supervised [78], and unsupervised methods are also in use [43]. More recently, researchers have developed methods for joint NER, entity typing and relation extraction [128]. We do not cover the breadth and intracacies of the numerous

approaches to relation extraction; instead, we refer the interested reader to the following surveys for more detail [95, 85].

2.1.3 Entity Resolution

After information extraction, the next step in knowledge base construction is often *entity resolution* (ER). ER is the task of *clustering* a collection of mentions by *ground-truth entity*. Ideally, each cluster contains *all* of the mentions of a single real-world entity. Consider Example 2, and assume that information extraction has produced a large collection of mentions—one per scientist—in BibTeX format. Assume that some of the mentions in the collection refer to a specific scientist named `Rajarshi Das`, who is affiliated with the organization `UMass Amherst`. The name *Rajarshi Das* is relatively common, and thus, it is reasonable to expect that there are multiple real-world scientists with this name who have corresponding mentions in the collection[11]. Making matters more challenging, some mentions may only include the first initial of the corresponding author's name (e.g., "R."), and some real-world authors may have multiple aliases. During ER, the goal is to build clusters of mentions such that each distinct, real-world, Rajarshi Das can be mapped to a single cluster with all of the corresponding mentions. We say that two mentions are **coreferent** if they refer to the same entity.

The challenge described above arises because of *identity uncertainty* [77, 84]. Formally,

Definition 1 (Identity Uncertainty)**.** *A mention exhibits **identity uncertainty** if there is uncertainty regarding the entity to which it refers.*

While identity uncertainty is well-known to be the primary difficulty in ER, in this thesis, we show that when a KB's underlying mentions exhibit identity uncertainty, user feedback provided to the KB *may also exhibit identity uncertainty* (Chapter 4). That is, in certain cases, the intended target (KB entity) of a piece of user feedback may be uncertainty. Indeed,

[11] This particular example is inspired by the DBLP profile page for "Rajarshi Das" which includes publications written by multiple distinct scientists with the same name.

this is one of the primary challenges of designing mechanisms for integrating user feedback and KB content.

As a crucial component of KB construction, ER has received significant attention from the research community [7, 91, 36, 86, 89, 99]. Related to Example 2 (and our experiments with OpenReview.net in Section 5.4), a large body of work focuses specifically on ER applied to mentions of scientific authors [42, 104, 21, 103, 120, 71, 110, 112, 127, 6]. Most of these approaches are comprised of: i) training a model of mention similarity, and ii) using the learned similarity metric in conjunction with a clustering algorithm to resolve the mentions. As is expected, no method perfectly solves ER, which adds to the potential of users to aid in KB content curation through feedback.

The ER studies referenced above are all set in the *cross-document* domain, where each mention may come from a different document. While we focus on cross-document ER in this thesis, for completeness, we also briefly discuss two related sub-tasks of KB construction: i) within-document coreference, and ii) entity linking. As the name suggests, within-document coreference (a.k.a., within-document ER) is the task of resolving mentions that all come from the same document. As such, within-document ER systems are generally more concerned with resolving pronouns, and cross-document ER typically involves a larger number of mentions and entities [99]. Unlike cross-document ER, the within-document setting allows for the use of an order-based approach known as *best-left-link* [83, 16]: during the resolution of a mention, that mention is either coreferent with a mention that appeared earlier in the document, or refers to an entity not yet referenced in the document. In comparison to the cross-document setting, where there is usually no ordering of the mentions, this drastically reduces the space of potentially coreferent mention pairs, thereby significantly lightening the computational burden. Indeed, many classic coreference resolution systems are founded on this approach [83, 11, 16]. Other systems take generative, graph-based, or, more recently, neural approaches [117, 40, 41, 122, 18, 19, 63]. For additional details, see the following survey of foundational related work [82].

Finally, we briefly describe entity linking. In this task, the input is a collection of mentions *and* an existing KB, and the goal is to link the mentions to their corresponding KB entity, or determine that their corresponding entity is not in the KB. Again, we refer the reader to surveys for further details on initial research [97] and newer, neural approaches [96]. In this thesis, we focus on ER (not entity linking) in the the cross-document setting. This is because we operate in a setting in which mentions come from multiple, heterogeneous sources, and no reference KB is initially available.

2.1.4 Canonicalization

After information extraction, relation extraction, and ER, it is often necessary to *canonicalize* the KB entities. At a high level, canonicalization refers to the process of synthesizing a single, consistent, and non-redundant "view" of a KB entity from its underlying mentions. For example, during canonicalization of open domain KBs—a setting in which canonicalization is particularly important—two mentions of the same entity, one expressing the relation `born in` and the other expressing the relation `birthplace`, would be combined to yield a single relationship. As another example, in the KB of scientists described above (Example 2), during canonicalization, the years and coauthors included in a collection of BIBTEX mentions of the scientist could be combined to produce a list of collaborators and years of collaboration. Compared to many other stages of KB construction, canonicalization has only received a small amount of attention by the research community. One study develops a method for resolving mentions—using clustering, much like ER—and then relationships using rule mining and clustering, followed by mapping the relation clusters to an external KB [33]. Another more recent works develops a joint approach to noun phrase and relationship canonicalization using embedded representations [107].

2.1.5 Knowledge Base Completion and Related Techniques

Once mentions are extracted and resolved, relationships discovered and entities canonicalized, the KB may be considered "built." Yet, at this stage, a KB has usually only realized

a small fraction of its potential utility. Often, additional methods are employed that leverage KB content and extends its functionality. Many of these techniques belong to the family of KB completion methods; all of these methods attempt to combining multiple, explicitly stored KB "facts" to infer missing entities, relationship and entity attributes. The diversity of work in this space is vast, and so we only raise a handful of examples.

In early systems, like Cyc, KB facts were often represented by logical predicates; logical inference rules could be applied to these facts to derive new facts and also answer questions [65]. The family of Universal Schema approaches represent a more recent approach to extending KB content that infer unobserved relationships via matrix factorization [93]. Different still is MINERVA, which learns to transverse a KB—represented as a graph—in order to answer queries about KB entities [22]. Today, some "KBs" are comprised of a corpus of natural language text—*not* highly structured facts, as in Cyc—coupled with a passage retrieval and a reading comprehension model [17, 23, 39, 87]. While these modern systems have been shown to support applications like question answering, in comparison to KBs like Cyc, they are less conducive to manual inspection and maintenance.

2.2 User Feedback for Improving KB Accuracy and Coverage

As a primary consumer of KBs, humans have the potential to improve their construction and maintenance. This thesis explores the intersection of user feedback and KB construction, focusing on KBs that exhibit identity uncertainty. Here, we briefly discuss the how user feedback could aid KB construction, the new challenges that arise, and how KB construction tools should be (re-)designed in response.

To begin, assume there exists a KB in which humans interact with the KB entities, *not* the underlying mentions. For example, in Example 2, users would interact with KB scientists, institutions, and canonicalized publications, but *not* the raw BibTeX records that are evidence of these entities. Interactions with the KB entities could include browsing

their corresponding wiki-style pages, issuing queries about entities and receiving responses generated by some form of KB inference, or other similar mechanisms.

During this interaction, humans may notice errors. Broadly, we identify four sources of KB errors:

1. missing data,
2. corrupted mentions, which are either the result of noisy raw data or information extraction,
3. mistakes made during ER, and
4. mistakes made during canonicalization, completion and other related tasks.

Since the users interact with KB entities, in all cases above, errors manifest similarly: they appear to users as KB entities with missing and spurious attributes. Upon encountering such an error, that user can submit feedback in order to correct it. To apply the feedback effectively, the KB must determine which source of error is responsible for the mistake, lest additional errors arise. As an example, consider a KB entity, e, with an incorrectly inferred email address, and user feedback conveying that e's email is incorrect. If the source of the error is noise or mistaken canonicalization, the KB should effectively use the feedback as evidence for removal of the incorrect email from e. On the other hand, if the source of the error is ER, then the KB must fix its underlying clustering of the mentions.

From this short example, a handful of desiderata are immediately apparent.

1. **Feedback must be provided at the entity-level.** This is because users interact with KB entities and not mentions. This is opposed to traditional mechanisms for providing feedback, which include pairwise mention constraints [113, 68, 108, 75, 76, 127].
2. **ER must be performed incrementally or online.** There are multiple reasons for such a design decision. First, in case the source of an error is faulty ER, a (partial) re-clustering of the mentions is required. Performing ER again from scratch may be

prohibitively expensive (especially if the KB supports real-time user feedback integration). Next, re-running a non-deterministic ER method may result in a completely different clustering of the mentions, which would also lead to very poor user experience. Finally, acquiring new data and user feedback naturally occurs continuously over time, which suggest a lightweight method of new data integration.

3. **ER must have the capability to split and merge KB entities.** This is required since feedback may be evidence of ER errors. This is in opposition of feedback integration schemes that simply overwrite or delete the attributes and relationships of KB entities.

4. **User feedback must be capable of facilitating recovery from all sources of error.** To this end, the way user feedback is represented in a KB is a critical design decision.

5. **User feedback should be treated *epistemologically* [121].** That is, user feedback should be treated as a first-class KB value; it should never be overwritten or deleted. This is because the KB may make mistakes when determining the source of an error that a piece of feedback attempts to remedy. We would like for the KB to be able to revisit these decisions in order to correctly apply the feedback.

6. **The KB must reason about identity uncertainty with respect to user feedback.** Since we assume that some errors arise from ER, it is possible for user feedback to have been applied to a KB entity whose mentions are later "split" to form two or more KB entities. In such cases, the KB must determine to which of the new KB entities the user feedback applies.

This thesis builds up a framework that aims to address each of these desiderata. To this end, the following chapters develop new algorithms, styles of user feedback and feedback representations. We provide both theoretical results where appropriate as well as empirical validation of our approach on both synthetic and real-world data.

CHAPTER 3

INCREMENTAL HIERARCHICAL CLUSTERING WITH TREE GRAFTING

3.1 Overview

Entity resolution (ER) is a central component of KB construction that is responsible for clustering mentions into inferred entities. The clustering of the KB's underlying mentions defines its entities and is used to derive their attributes and relationships. Unfortunately, ER algorithms are inevitably imperfect; they often cluster mentions from different ground-truth entities together and split mentions from the same ground-truth entity across multiple clusters. These errors manifest after canonicalization as spurious and missing entity attributes and relationships.

Fortunately, users can help to identify these ER errors through the contribution of feedback, which we also refer to as *user edits*. Ideally, a KB would be equipped with an algorithm capable of consuming the user edits to incite mergers and splits among the previously inferred entities with the goal of discovering the ground-truth partition. Since these user edits naturally arrives over time, ideal integration algorithms are *incremental*, i.e., they consume one piece of data at a time rather than reclustering all of the mentions after each new piece of feedback arrives.

In this chapter, we develop GRINCH, the **G**rafting and **R**otation-based **INC**remental **H**ierarchical clustering algorithm that can be used for ER. GRINCH consumes points one at a time and adds them to an incrementally built hierarchy. Unlike previous incremental clustering algorithms, GRINCH includes a deterministic, *global* rearrangement mechanism called a `graft`. During a `graft`, two subtrees that may be far from each other in the tree

can be merged if they are deemed similar enough. This type of global rearrangement is imperative for appropriately handling user edits (and new mentions) that provide evidence for the merger of two distinct inferred entities. Unlike many clustering algorithms, GRINCH can be used to cluster data according to any similarity model, including *linkage functions*, which measure similarity between two subtrees.

Our choice to develop a hierarchical clustering algorithm is partially inspired by extensive work in hierarchical models and algorithms for ER [21, 103, 99, 120, 66, 71, 127, 6]. In this work, learned linkage functions have led to improvements in accuracy by helping to identify set-level inconsistencies and similarities with respect to attributes like: gender, animacy and number (singular/plural) [88, 28, 19]. Additionally, hierarchical models promote efficiency in inference and facilitate the representation of uncertainty by encoding multiple partitions of the underlying mentions simultaneously [126, 31, 55].

Theoretically, we define a notion of *model-based separation* that characterizes the relationship between a linkage function and a dataset. For generality, we adopt a graph-theoretic formalism, where data points correspond to vertices of an unknown graph whose connected components form a ground-truth clustering. Model-based separation suggests that the linkage function value is high for two item sets if the induced subgraph is connected (see Subsection 3.2.1). We prove that under this condition, the ground-truth clusters are a tree-consistent partition of the hierarchy built by GRINCH.

In experiments, we show that GRINCH is efficient and builds trees with higher dendrogram purity than other clustering algorithms on large scale datasets. The experiments are performed with a common and important linkage function—average linkage—as well as a linkage function that measures the cosine similarity between two cluster centroid representations. We also perform experiments on two author coreference (a.k.a., author ER) datasets using learned linkage functions, and demonstrate that GRINCH is more efficient and accurate than the baselines. Our experiments reveal that GRINCH dominates competitors

that only make local tree rearrangements, highlighting the power of the `graft` subroutine and the robustness of GRINCH.

3.2 Linkage Functions for Clustering

Clustering is the problem of constructing a partition $\mathcal{C} = \{C_1, \cdots, C_k\}$ of a dataset $\mathcal{X} = \{x_i\}_{i=1}^N$, such that $\bigcup_{C \in \mathcal{C}} C = \mathcal{X}$ and $\forall C, C' \in \mathcal{C}, C \cap C' = \emptyset$. The partition is known as a *clustering* of $\mathcal{X}$.

Most algorithms construct clusterings using pairwise similarities among points. But, pairwise similarities cannot capture many complex relationships, e.g., points x_1 and x_2 are similar when clustered with point x_3, but are otherwise dissimilar. A natural generalization that can capture these types of relationships are similarities defined over sets of points, which we refer to as linkage functions. Formally, a linkage function is a function $f : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \to \mathbb{R}$.

Clustering with linkage functions is ubiquitous, especially in HAC (from which the name linkage function is derived). In HAC, many popular linkage functions like single-, complete- and average-linkage are computed from pairwise distance functions. More complex, set-wise linkage functions are used in applications such as image segmentation, within document coreference and entity resolution; in the latter two domains, these functions are often learned [40, 19, 58, 41, 122, 125]. A unique capability of HAC is that it can easily support an arbitrary linkage function. This flexibility is essential to combat the ill-posed nature of clustering.

3.2.1 Model-based Separation

Our goal is to design an algorithm that, like HAC, can support arbitrary linkage functions, but is dramatically faster. In developing clustering algorithms, it is often useful to consider various assumptions about the *separability* of the underlying data. For example, in the pairwise setting one of the strongest data assumptions is known as *strict separation* [8]. This assumption holds that any point in ground-truth cluster C_i is more similar to every other point

in C_i than any point from a different ground-truth cluster, C_j. It is easy to see that popular instantiations of HAC (e.g., single-, average- and complete-linkage) provably succeed under strict separation, which provides some theoretical motivation for these algorithms.

We introduce a notion of *model-based separation* for clustering with a linkage function. Since linkage functions may operate on data of any type, we formalize the definition in terms of a graph, where the points correspond to vertices.

Definition 2 (Model-based Separation)**.** *Let $G = (\mathcal{X}, E)$ be a graph. Let $f : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \to \mathbb{R}$ be a linkage function that computes the similarity of two groups of vertices and let $\phi : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \to \{0, 1\}$ be a function that returns 1 if the union of its arguments is a connected subgraph of G. Then f separates G if*

$$\forall s_0, s_1, s_2 \subseteq \mathcal{X}, \quad \phi(s_0, s_1) > \phi(s_0, s_2) \Rightarrow f(s_0, s_1) > f(s_0, s_2)$$

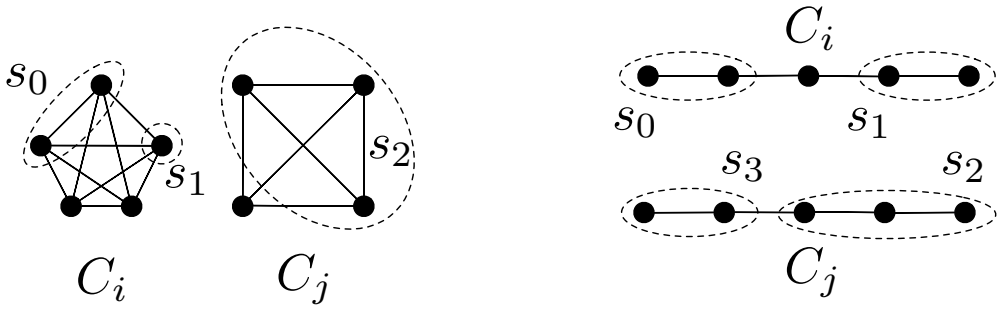
In words, for a linkage function f to separate a graph G, take any two sets of vertices, s_0 and s_1, such that $s_0 \cup s_1$ is connected in G, i.e., $\phi(s_0, s_1) = 1$. Then, for any set s_2 such $\phi(s_0, s_2) = 0$, the score of f on input (s_0, s_1) must be greater than on input (s_0, s_2).

Model-based separation offers a non-standard view of clustering. Specifically, the points of a dataset are treated as vertices in a graph with latent edges. The ground-truth clusters are the connected components of the graph and the goal of clustering is to discover these components using a linkage function.

We provide the following two examples to help build intuition about model-based separation. The examples are used throughout the remainder of our discussion.

Example 3 (Clique)**.** *Consider a graph $G = (\mathcal{X}, E)$ in which each connected component is a clique. Then if f separates G, every vertex in a connected component, C_i, is more similar to all other vertices in C_i than any vertex in connected component C_j, where similarity is defined by f.*

Thus, clique-structured connected components exactly capture strict separation.



(a) Clique-shaped clusters.

(b) Chain-shaped clusters.

Figure 3.1: *Model-based separation*. Figure 3.1a shows two clique-shaped clusters with points as vertices in a graph. If f separates the graph then $f(s_0, s_1) > \max[f(s_0, s_2), f(s_1, s_2)]$ because s_0 and s_1 form a connected subgraph. In Figure 3.1b, even if f separates the graph, it is possible for $f(s_0, s_1) < f(s_1, s_2)$. However, $f(s_1, s_2) < f(s_2, s_3)$.

Example 4 (Chain)**.** *Consider a graph $G = (\mathcal{X}, E)$ in which each connected component is chain-structured. According to Definition 2, two vertices that are part of the same chain but do not share an edge may be dissimilar under f even if f separates G. However, any two segments of the chain connected by an edge are similar under f.*

A visual illustration of both clique and chain style clusters is depicted in Figure 3.1. As we will see, chain structured connected components pos a challenge to existing incremental algorithms, something we resolve with GRINCH (Section 3.3).

3.2.2 Cluster Trees

In most clustering problems, the appropriate number of clusters is unknown a priori. HAC addresses this uncertainty by building a *cluster tree* over points.

Definition 3 (Cluster tree [59])**.** *A binary **cluster tree** $\mathcal{T}$ on a dataset $\mathcal{X} = \{x_i\}_{i=1}^N$ is a collection of subsets such that $C_0 = \{x_i\}_{i=1}^N \in \mathcal{T}$ and for each $C_i, C_j \in \mathcal{T}$ either $C_i \subset C_j$, $C_j \subset C_i$ or $C_i \cap C_j = \emptyset$. For any $C \in \mathcal{T}$, if $\exists C' \in \mathcal{T}$ with $C' \subset C$, then there exists two $C_L, C_R \in \mathcal{T}$ that partition C.*

Given a cluster tree, $\mathcal{T}$, any set of disjoint subtrees whose leaves cover $\mathcal{X}$ represents a valid clustering and is referred to as a *tree consistent partition* [45]. Thus, cluster trees compactly encode multiple alternative clusterings, allowing for a clustering to be selected as a post-processing step. Another advantage of using cluster trees is that they often facilitate efficient search and naturally group similar points near one another in the hierarchy

We relate model-based separation, cluster trees and HAC in the following fact:

Fact 1. *Let f be a linkage function that separates G. Then running* HAC *under f returns a cluster tree, $\mathcal{T}$, such that the connected components of G are a tree-consistent partition of $\mathcal{T}$.*

To see why, notice that in each iteration of HAC, the highest scoring pair of remaining subtrees is merged. Since f separates G, a merger resulting in a subtree that corresponds to a connected subgraph of G has higher score than any merger resulting in a disconnected subgraph of G. Even though HAC can construct a cluster tree that contains the ground-truth clustering as a tree-consistent partition, the algorithm costs $O(n^2 \log n)$ for general linkage functions and does not scale to large datasets. We will verify this claim empirically in our experiments (Section 3.4).

3.3 Rotations, Grafting and Grinch

In this section we derive an efficient, incremental algorithm called GRINCH that can be used to construct clusterings under any linkage function. Like HAC, the backbone of GRINCH is a cluster tree. We begin the discussion by analyzing a greedy, incremental variant of HAC and when it fails. Then, we introduce two subroutines, `rotate` and `graft`, that can be used to enhance robustness. Finally, we present our algorithm, GRINCH.

Our discussion of GRINCH, includes notation relating various cluster-tree nodes to one another. Figure 3.2 visualizes these relationships.

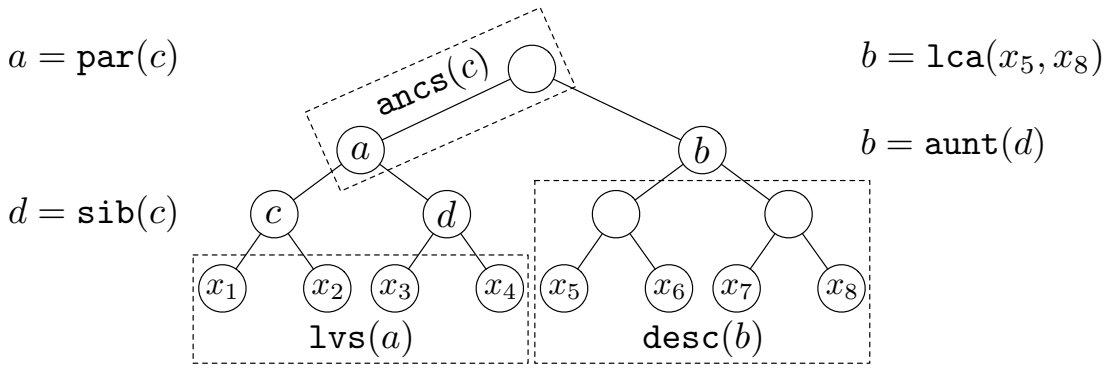


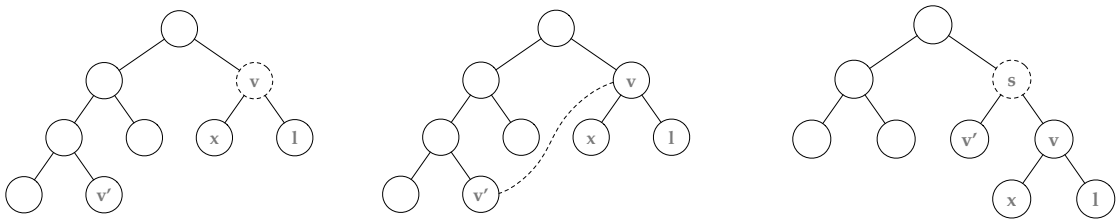
Figure 3.2: **Tree Terminology**. a is the parent of c and d is the sibling of c. b is the aunt of d and also the least common ancestor (LCA) of x_5 and x_8. c has two ancestors, a has four leaves, and b has six descendants.

3.3.1 Online HAC and Rotations

An efficient alternative to HAC is its online variant that merges each incoming point with its nearest neighbor seen so far (GREEDY). For now, let us consider the setting in which a nearest neighbor is found using a linkage function, f. Let f separate a graph G and let ground-truth clusters be cliques in G (i.e., the data is strictly separated). Even in this simple case, GREEDY may construct a cluster tree in which the ground-truth clustering is not a tree consistent partition. To see why, consider a stream in which the first two points, x_1 and x_2, are of the same ground-truth cluster and the third point, x_3 is of a different cluster. Assume, without loss of generality, that GREEDY adds x_3 as a sibling of x_1. Then the ground-truth clustering is not a tree consistent partition of the resulting tree (and all subsequent trees).

To recover from such mistakes, local tree rearrangements may be applied. Previous work uses *rotations*, which swap a child and its aunt in the tree, to correct local errors induced by unfavorable arrival order [55]. While originally designed to be used with pairwise distances, the condition under which rotations should be applied can be extended to linkage functions:

$$f(v, \texttt{sib}(v)) < f(v, \texttt{aunt}(v)) \tag{3.1}$$



(a) x is added to the cluster tree. (b) v finds nearest neighbor v'. (c) v' is grafted to v.

Figure 3.3: The `graft` subroutine. Dotted lines denote new nodes and mergers. Before x is added to tree, l and v' reside in disjoint subtrees even though they belong to the same ground-truth cluster. The addition of x creates the subtree with root v and initiates the `graft` subroutine.

where the functions $\texttt{sib}(\cdot)$ and $\texttt{aunt}(\cdot)$ return the sibling and aunt of their input, respectively. In words, if a node $v \in \mathcal{T}$ achieves a higher score under f with its aunt than with its sibling, then the aunt and sibling should be swapped. Now, let us revisit the example above. Since x_1 and x_2 are both vertices in the same clique in G, they are connected by an edge. Then, by model-based separation, $f(x_1, x_2) > f(x_1, x_3)$, so a rotation will be applied, producing a tree that contains the ground-truth clustering.

Unfortunately, the GREEDY algorithm, augmented with the ability to performs rotations (ROTATE), cannot always recover the connected components of a graph that is separated by f. In particular, ROTATE cannot reliably recover chains (Example 4). By virtue of being a local operation, rotations can only be used to provably recover connected components that are clique-structure.

3.3.2 Subtree Grafting

We introduce a non-local tree rearrangment called a *graft*, which facilitates the discovery of chain-structured connected components. At a high level, the `graft` procedure with respect to a node $v \in \mathcal{T}$ searches $\mathcal{T}$ for a node v' that is both similar to v and dissimilar from its current sibling, $\texttt{sib}(v')$. If such a subtree is found, v' is disconnected from its parent and made a sibling of v. A visual illustration of a successful `graft` is depicted in Figure 3.3.

In detail, a `graft` searches the leaves of $\mathcal{T}$ for the nearest neighbor leaf of v called l. Then it checks whether the following holds:

$$f(v, l) > \max[f(v, \texttt{sib}(v)), f(l, \texttt{sib}(l))] \tag{3.2}$$

i.e., v and l prefer each other to their current siblings according to f. If the condition succeeds, merge v and l. If the condition fails because l prefers its sibling to v, retest the condition at v and l's parent, $\texttt{par}(l)$; if the condition fails because v prefers its sibling to l, then retest the condition at $\texttt{par}(v)$ and l. Continue to check recursively until the condition succeeds or until the first time two nodes, v_1 and v_2, are reached such that one is the ancestor of the other. Pseudocode for the `graft` subroutine can be found in Algorithm 1. In the algorithm, `par` returns the parent of a node in the tree, `lca` returns the lowest common ancestors of its arguments and `makeSib` merges its arguments and returns their new parent. `NN` performs a nearest neighbor search and `constrNN` performs a nearest neighbor search that excludes its second argument from the result.

3.3.3 Tree Restructuring

While the `graft` subroutine facilitates discovery of chain-structured clusters, poorly structured trees are susceptible to having the `graft` subroutine disconnect previously discovered ground-truth clusters. As an example, consider Figure 3.4, in which $\texttt{lvs}(v)$ form the connected subgraph C_i (i.e., they all belong to the same ground-truth cluster). Consider v's left child, $v.l$, and its descendants, which form a *disconnected* subgraph. An attempt to `graft` either descendent, x_1 or x_2, may succeed, even when initiated from a node (not depicted) whose descendants are not connected to C_i. After such a `graft`, $\mathcal{T}$ cannot contain a tree-consistent partition that matches the ground-truth clustering.

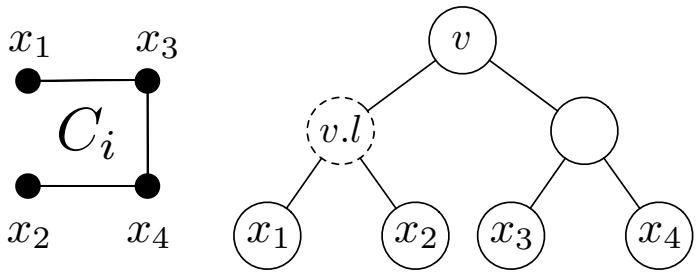


Figure 3.4: *Poorly structured tree*. Even though v's leaves form a connected subgraph of the graph on the left of the Figure (i.e., they all belong to cluster C_i), $v.l$'s descendent leaves, x_1 and x_2, are a disconnected. An attempt to `graft` either x_1 or x_2 from a node whose descendants are not members of C_i may succeed.

Algorithm 1 `graft` $(v, \mathcal{T}, f)$

```
l = constrNN(v, lvs(v), f, T)
v' = lca(v, l); st = v
while v ≠ v' ∧ l ≠ v' ∧ sib(v) ≠ l do
  if f(v, l) > max[f(v, sib(v)), f(l, sib(l))] then
    z = sib(v); v = merge(v, l)
    restruct(z, lca(z, v), f)
    break
  if f(v, l) < f(l, sib(l)) then
    l = par(l)
  if f(v, l) < f(v, sib(v)) then
    v = par(v)
if v == st then
  Output: v'
else
  Output: v
```

Notice that a subtree can defend against spurious `grafts` by ensuring that each of its descendant subtrees is connected. For example, in Figure 3.4, if x_2 and x_3 were swapped, then each descendant subtree of v would be connected. Moreover, after such a swap, `grafts` from nodes whose descendants were not part of C_i would necessarily fail (assuming that f separates the graph).

During tree construction, the only step that can result in a connected subtree with disconnected descendants is the `graft` subroutine (a rigorous proof is included in the appendix). We introduce the `restruct` (*restructure*) subroutine, which is performed

Algorithm 2 $\texttt{restruct}(z, r, f)$

while $z! = r$ **do**
  $as = \{\texttt{sib}(a) \texttt{ for } a \in \texttt{ancs}(z) \backslash \texttt{ancs}(r)\}$
  $m = \text{argmax}_{a \in as} f(z, a)$
  if $f(z, \texttt{sib}(z)) < f(z, m)$ **then**
    $\texttt{swap}(\texttt{sib}(z), m)$
  $z = \texttt{par}(z)$

after a successful `graft`, and reorganizes a subtree with the intent of making each of its descendants connected. Let v' be a node that was just grafted, v be the previous sibling of v' (i.e., before the graft) and let $r = \texttt{lca}(v, v')$ be the current least common ancestor of v and v'. `restruct` is initiated from v. First, the siblings of the ancestors of v (until r) are collected. Then, we find the node in the collection most similar to v. If that node is more similar to v than v's current sibling (according to f), the two are swapped. The intuition here is that if a `graft` left v and its new sibling disconnected, then the swap serves as a mechanism to restore the connectedness of v's parent. Such swaps are attempted from the ancestors of v until r. Pseudocode appears in Algorithm 2.

3.3.4 Grinch

Using the `rotate`, `graft` and `restruct` tree rearrangement routines discussed in Section 3.3, we derive a new algorithm called GRINCH, which stands for: **G**rafting and **R**otation-based **INC**remental **H**iearchical clustering. The steps of the algorithm are as follows: when a new record, x_i, arrives, find x_i's nearest neighbor, l, among the leaves of $\mathcal{T}$. Add x_i to $\mathcal{T}$ as a sibling of l. Then, apply the `rotate` subroutine while Equation 3.1 is true. Finally, attempt to `graft` recursively from each ancestor of x_i. Each time a `graft` is successful, restructure the tree to group similar items together. Pseudocode for GRINCH can be found in Algorithm 3.

Theorem 1. *Let $\mathcal{X} = \{x_i\}_{i=1}^N$ be a dataset with ground-truth clustering $\mathcal{C}^\star = \{C_1, \cdots, C_k\}$. Let f separate a graph G on vertices $\mathcal{X}$ and let each cluster $C \in \mathcal{C}^\star$ be a connected*

Algorithm 3 `Insert`$(x_i, \mathcal{T}, f)$

$l =$ `NN`$(x_i, f, \mathcal{T})$; $t =$ `makeSib`(x_i, l)
while $f(x_i, \texttt{sib}(x_i)) < f(\texttt{aunt}(x_i), \texttt{sib}(x_i))$ **do**
  `rotate` $(x_i, \texttt{aunt}(x_i))$
$p = \texttt{par}(x_i)$
while $p \neq \texttt{null}$ **do**
  $\texttt{curr} = \texttt{graft}(p, \mathcal{T}, f)$

component in G. Then GRINCH *recovers a cluster tree such that $\mathcal{C}^\star$ is a tree consistent partition of $\mathcal{T}$ regardless of the input order.*

The proof of Theorem 1 can be found in the appendix.

3.4 Experiments

We experiment with GRINCH to assess its scalability and accuracy. We begin by demonstrating that GRINCH outperforms other incremental clustering algorithms on a synthetic dataset. Observing that some of the steps of GRINCH are underutilized, we present 4 approximations of GRINCH's algorithmic components. We apply each approximation in turn and show that together they dramatically improve GRINCH's scalability without compromising its clustering quality. Then, we compare the approximate variant of GRINCH to state-of-the-art large scale hierarchical clustering methods. To showcase the flexibility of GRINCH, we also provide experimental results in entity resolution, where the linkage function is learned. Finally, we provide analysis of the `graft` subroutine–GRINCH's distinguishing feature–and perform experiments to demonstrate the algorithm's robustness.

3.4.0.0.1 Dendrogram Purity Before beginning, we briefly review *dendrogram purity*, a preferred method of holistically evaluating hierarchical clusterings [13, 45, 55]. Dendrogram purity is computed as follows: Let $\mathcal{C}^\star = \{C_1, \cdots, C_k\}$ be the ground-truth clustering of a dataset $\mathcal{X}$, and let $\mathcal{P}^\star = \{(x, x') | x, x' \in \mathcal{X},\ \mathcal{C}^\star(x) = \mathcal{C}^\star(x')\}$ be the set of all point pairs that belong to the same ground-truth clusters. Then the dendrogram purity (DP) of

a cluster tree, $\mathcal{T}$ is: where $\texttt{lca}(x, x')$ returns the least common ancestor of x and x' in $\mathcal{T}$, $\texttt{lvs}(\cdot)$ returns the descendant leaves of its argument, and $\texttt{pur}(\cdot, \mathcal{C}^\star(x),)$ takes a collection of leaves and computes the fraction that belong to ground-truth cluster $\mathcal{C}^\star(x)$.

3.4.1 Synthetic Data Experiment

In our first experiment, we compare GRINCH to other incremental hierarchical clustering algorithms on a synthetic dataset in order to begin to understand GRINCH's empirical performance characteristics in a controlled manner. The data is generated so that it satisfies model-based separation with respect to cosine similarity. In particular, the dataset contains 2500 10000-dimensional binary vectors that belong to 100 clusters, with 25 points per cluster. Points in cluster k have bits $100k$ to $100(k-1)$ set randomly to 1 with probability 0.1. All other bits are set to 0. This way, across cluster points have cosine similarity 0 and within cluster points can have either 0 or non-zero cosine similarity. In other words, two points, x_1 and x_2, in the same cluster can appear to be dissimilar and end up in distant regions of the tree. The representation of each internal node in the GRINCH tree is the sum of the vectors of its descendent leaves. Thus, compute the cosine similarity between two nodes v and v' as the cosine similarity between their aggregated vectors (we refer to this as *cosine linkage* in the following sections). We compare GRINCH, ROTATE and GREEDY.

The experimental results reveal that GRINCH achieves perfect dendrogram purity (1.0), which is expected given GRINCH's correctness guarantee. ROTATE achieves a dendrogram purity of 0.872 while GREEDY achieves 0.854. ROTATE and GREEDY do not construct trees of perfect purity because of their inability to globally rearrange a cluster hierarchy.

3.4.2 Approximations

Some of the algorithmic steps of GRINCH, which are required to prove its correctness, are seldom invoked in practice. For example, and perhaps expectedly, a `graft` is unlikely to succeed between two nodes close to the root of the tree. Therefore, we introduce handful

of approximations designed to have little effect on the quality of the clusterings constructed by GRINCH, but also designed to make the algorithm significantly faster in practice.

1. **Capping.** Recursive subroutines like `graft` and `rotate` improve performance, but they are also computationally expensive to check, and often fail. Moreover, we notice that tree rearrangements that occur close to the root do not have a significant, instantaneous effect on dendrogram purity. Therefore, we introduce *rotation*, *graft* and *restructure caps*, which prohibit rotations, grafts and restructures from occurring above a height, h.
2. **Single Elimination Mode.** The `graft` subroutine generally improves GRINCH's clustering performance, and is essential in attaining perfect purity on the synthetic dataset, but we find that `graft` attempts are rejected many more times than they are accepted. However, at times, we observe that a sequence of recursive `grafts` are accepted when initiated close to the leaves. Therefore, to limit the number of attempted `grafts` while retaining these `graft` sequences, we introduce *single elimination mode*. In this mode, the recursive grafting procedure terminates after a `graft` between v and v' fails because both prefer their current siblings to a merge.
3. **Single Nearest Neighbor Searching.** GRINCH makes heavy use of nearest neighbor search under the linkage function f. Rather than perform nearest neighbor search anew for each `graft`, when a point arrives, we perform a single k-NN search ($k \in [25, 50]$) and only consider these nodes during subsequent `grafts` (until the next point arrives).
4. **Navigable Small World Graphs.** Instead of performing nearest neighbor computations exactly, we can perform them approximately. To this end, we employ a *navigable small world* nearest neighbor graph (NSW)–a data structure inspired by decentralized search in small world networks [116, 53, 54]. To find the nearest neighbor of a point, x_i, in an NSW, begin at a random node, v. If the similarity between x_i and v is maximal among all neighbors of v, terminate; otherwise, move to the neighbor of v most similar to x_i. To insert a new point, x_j, find its k nearest neighbors and add edges between those neighbors

	ALOI					Synthetic				
Approx.	**DP**	**Time (s)**	**# Rotate**	**# Graft**	**# Restr.**	**DP**	**Time (s)**	**# Rotate**	**# Graft**	**# Restr.**
GRINCH	0.533	85.37	7107	2435	1088	1.0	160.31	2558	578	203
+Cap (100)	0.533	48.45	6495	2157	686	0.993	164.33	2558	578	194
+1-Elim.	0.534	39.02	6574	1586	533	0.997	157.62	2523	526	184
+1-NN	0.540	22.23	6441	1516	570	0.993	83.01	2517	415	148
+no Restr.	0.538	14.29	6477	1634	0	0.993	82.26	2476	426	0
+no Graft	0.506	12.75	6747	0	0	0.872	82.06	2259	0	0
+no Rotate	0.442	14.79	0	0	0	0.854	80.53	0	0	0

Table 3.1: *Ablation.* Each row in the table represents GRINCH with the corresponding approximation applied in addition to all approximations contained in previous rows. The first 4 approximations significantly decreases the computational cost of GRINCH, but do not compromise DP. The ablation is performed for the first 5000 points of ALOI and the Synthetic datasets.

and a new point [73]. Thus, NSWs are constructed online. In practice, we simultaneous construct a hierarchical clustering and an NSW over the points stored in the tree's leaves.

To measure the effects of our approximations on the speed and quality of the resulting algorithm, we conduct the following ablation. We run GRINCH on our synthetically generated dataset as well as a random 5k subset of the ALOI [35] dataset and measure dendrogram purity, time, and the number of calls made to `rotate`, `graft` and `restruct`. We repeat the procedure multiple times, each time adding one of the following approximations, in order: capping, single elimination, single nearest neighbor search and approximate nearest neighbor search. Capping and is performed at height 100. We also experiment with removal of the `graft` and `rotate` subroutines.

The result of the ablation is contained in Table 3.1. We observe that, for both datasets, each of the approximations reduces the computational cost of algorithm without effecting the resulting DP. However, once `grafts` are removed, the DP drops by 3% on ALOI and 12% on the synthetic datasets. When `rotate` is also removed, DP drops by an additional 6% and 2%, respectively.

Having verified that on a subset of ALOI our approximations improve scalability at little expense in terms of dendrogram purity, in the following experiments we report results for GRINCH in single elimination mode and with the rotation cap set to $h = 100$.

Algorithm	Linkage	CovType	ILSVRC12 (50k)	ALOI	Speaker	ImageNet (100k)
GRINCH	Avg	0.43 ± 0.00	$\mathbf{0.557 \pm 0.003}$	$\mathbf{0.504 \pm 0.002}$	0.480 ± 0.003	$\mathbf{0.065 \pm 0.000}$
GRINCH	CS	0.43 ± 0.00	$\mathbf{0.544 \pm 0.005}$	$\mathbf{0.499 \pm 0.003}$	0.478 ± 0.003	0.062 ± 0.000
ROTATE	Avg	0.43 ± 0.01	0.545 ± 0.004	0.476 ± 0.004	0.407 ± 0.003	0.063 ± 0.001
ROTATE	CS	0.44 ± 0.01	0.513 ± 0.007	0.472 ± 0.003	0.406 ± 0.003	0.062 ± 0.000
GREEDY	—	0.44 ± 0.01	0.527 ± 0.004	0.435 ± 0.004	0.317 ± 0.002	0.0589
PERCH [55]	—	0.45 ± 0.004	0.53 ± 0.003	0.44 ± 0.004	0.37 ± 0.002	$\mathbf{0.065 \pm 0.000}$
PERCH-BC [55]	—	0.45 ± 0.004	0.36 ± 0.005	0.37 ± 0.008	0.09 ± 0.001	0.03 ± 0.00
MB-HAC [55]	Best Reported	0.44 ± 0.005	0.43 ± 0.005	0.30 ± 0.002	0.01 ± 0.002	—
HAC [55]	Average	–	**0.54**	–	**0.55**	–

Table 3.2: Dendrogram Purity results for GRINCH and baseline methods. We compare two linkage functions: approximate average linkage (Avg) and cosine similarity linkage (CS).

3.4.3 Large Scale Clustering

We compare GRINCH with the following 4 algorithms: **GREEDY** - an online hierarchical clustering algorithm that consumes one point at a time and places it as a sibling of its nearest neighbor; **ROTATE** - an incremental algorithm that places a point next to its nearest neighbor and then performs rotations until Equation 3.1 holds; **MB-HAC** - the mini-batch version of HAC, which keeps a buffer of size b, runs a single step of HAC using the points in the buffer and then adds the next record to the buffer; **HAC** - best-first, bottom-up hierarchical agglomerative clustering and **PERCH** - a state-of-the-art large scale hierarchical clustering method.

We run each algorithm on 5 large scale clustering datasets: CovType, a datset of forest covertype, ALOI [35], a 50K subset of the Imagenet ILSVRC12 dataset [94] and the Speaker dataset [37], and a 100K subset of ImageNet containing all 17K classes not just the subset in ILSVRC12. Datasets have 500K, 50K, 100K, 36K, and 100K instances, respectively. We run each HAC variant under two different linkage functions: average linkage and cosine linkage. To compute the cosine similarity between two nodes, v and v', first, for each node, compute the sum of the vectors contained at their descendant leaves. Then, compute the cosine similarity between the aggregated vectors.

Results are displayed in Table 3.2, where we record the dendrogram purity averaged over 5 replicates of each algorithm, where for each replicate we randomize the arrival order of the data. The table reveals that GRINCH–under both linkage functions–outperforms

the corresponding versions of ROTATE and GREEDY on all datasets except for on the CovType dataset where the methods all seem to perform equally well. This underscores the power of the `graft` subroutine. GRINCH with approximate nearest neighbor search even outperforms PERCH, which uses exact nearest neighbor search, on ALOI. Recall that, unlike the HAC variants, PERCH employs a specific linkage function. Seeing as the HAC variants outperform PERCH on Speaker suggests that the ability to equip various linkage functions can be advantageous. HAC is best on Speaker, but cannot scale to ALOI.

3.4.4 Author Coreference

Bibliographic databases, like PubMed, DBLP, and Google Scholar, contain citation records that must be attributed to the corresponding authors. For some records, the attribution process is easy, but for many others, the identities of a publication's authors are ambiguous. For example, DBLP contains hundreds of citations written by different authors named "Wei Wang" that currently cannot be disambiguated [1]. Intuitively, author coreference datasets often exhibit chain like structures because a single citation written by a prolific author (perhaps in a short-lived collaboration) may only be similar to a small number of that author's other citations and dissimilar from the rest.

Following previous work, we train a linkage function to predict the likelihood that a group of citation records were all written by the same author [21, 99, 120]. We train our model by running HAC and, at each step, use the model to predict the precision of merging two groups of records. (A similar training technique was previously proposed for entity and event coreference [62].) Our model has access to features like: coauthor names and publication title, venue, year, etc.

We compare the 5 HAC variants in author coreference on two datasets with labeled author identities: **Rexa** [21] and **PSU-DBLP** [42]. As is standard in author coreference we evaluate the methods using the pairwise F1-score of a predicted flat clustering against the ground-truth clustering, which is the harmonic mean of precision and recall. To compute

	Rexa			DBLP		
Algorithm	**Pre**	**Rec**	**F**	**Pre**	**Rec**	**F**
GRINCH	0.808	0.883	**0.844 ± 0.004**	0.809	0.620	**0.701 ± 0.013**
ROTATE	0.864	0.641	0.734 ± 0.057	0.876	0.554	0.678 ± 0.019
GREEDY	0.850	0.209	0.331 ± 0.094	0.827	0.151	0.255 ± 0.027
MB-HAC-Med.	0.807	0.881	**0.843 ± 0.0009**	0.375	0.631	0.461 ± 0.072
MB-HAC-Sm.	0.922	0.333	0.483 ± 0.061	0.697	0.151	0.247 ± 0.004
EXACT	0.805	0.887	**0.844**	0.741	0.600	0.664

Table 3.3: Precision, recall and F-Score of various methods on the Rexa and DBLP datasets.

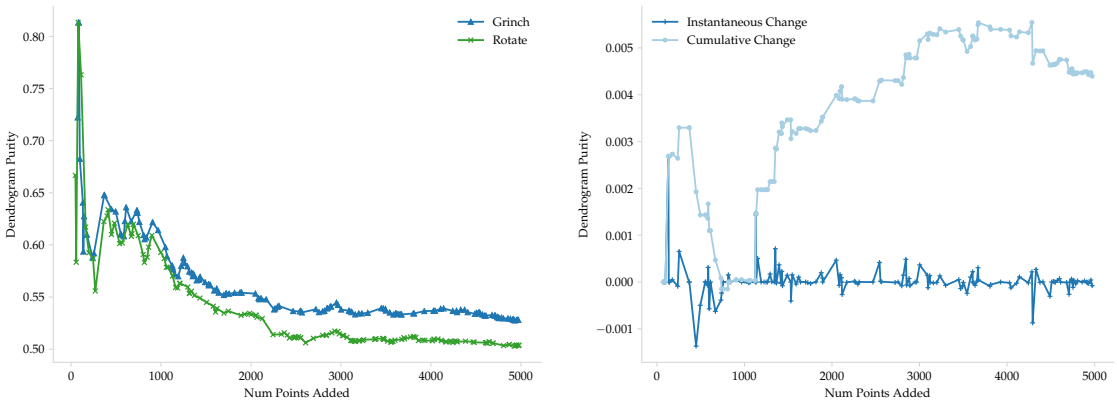
pairwise F1-score, each pair of citations that appear in both the same ground-truth and predicted clusters is considered a true positive; each pair of citations that belong to different ground-truth clusters but the same predicted cluster is considered a false positive. None of the authors represented in the test set, have any publications in the training set.

Figure 3.3 shows the precision, recall, and pairwise F1-score achieved by each method. The results show that GRINCH outperforms the other scalable methods on both datasets and even outperforms HAC on DBLP. This behavior may stem from overfitting of the learned linkage function, which is exploited by HAC; since GRINCH only approximates HAC, it can be thought of as a form of regularization.

Again, we observe that GRINCH outperforms GREEDY and ROTATE on both datasets underscoring the importance of the `rotate` and `graft` procedures.

3.4.5 Significance of Grafting

The results above indicate that GRINCH–even when employing a number of approximations–constructs trees with higher dendrogram purity than other scalable methods in a comparable amount of time. Interestingly, GRINCH only differs from `rotate` in its use of the `graft` (and subsequent `restruct`) subroutine. To better understand the significance of grafting, we compare GRINCH and `rotate` on the first 5000 points of ALOI.



(a) Dendrogram purity per point.

(b) Change in DP due to `grafts`.

Figure 3.5: Figure 3.5a shows the dendrogram purity of two trees, one built by GRINCH and the other built by `rotate`, on the first 5000 points of ALOI. The dendrogram purity of the tree built GRINCH is greater than that of the tree built by `rotate`. Figure 3.5b plots the instantaneous and cumulative change in dendrogram purity due to `grafts`. While GRINCH achieves 3% larger dendrogram purity than `rotate`.

Figure 3.5a shows that dendrogram purity as a function of the number of data points inserted for both GRINCH and `rotate` and the first 5000 points of ALOI. Echoing the results above, by 1000 points, GRINCH dominates `rotate`.

Figure 3.5b shows the instantaneous and cumulative change in dendrogram purity due to `grafts` made by GRINCH. That is, for the ith data point, x_i, we record the dendrogram purity after x_i is inserted and rotations are performed (i.e., what would be executed by `rotate`). Then, we perform `grafting` (if appropriate) and record the dendrogram purity *after* all recursive `grafts` have been completed. The difference between the dendrogram purity after `grafting` and before `grafting` (but after rotations) is the instantaneous change in dendrogram purity due to `grafts`; the sum of instantaneous changes is the cumulative change.

Note the y-axis of Figure 3.5b, which reveals that even the most instantaneously significant `grafts` only lead to minute changes in dendrogram purity (of about 0.001). Moreover, after 5000 points, the cumulative change in dendrogram purity due to `grafts` is less than

0.005–hardly accounting for the difference in dendrogram purity between the tree built by GRINCH and the tree built by `rotate` (of 0.03). We conclude from these measurements that the increase in performance due to the `graft` subroutine is related to the rearrangement of small numbers of points. These rearrangements do not immediately have significant impact on dendrogram purity, but they do have significant long-term affects. To make this hypothesis more concrete, consider the case in which a two dissimilar date points from the same cluster are split between two distant regions of the tree early on in clustering. The points are never merged (via a `graft`) and so each point draws a significant portion of the cluster's other points to its location in the tree. This has dire consequences with respect to dendrogram purity. If a `graft` is performed early on to correct the split, an adverse scenario like this can be averted.

3.4.6 Robustness

For completeness, we perform an experiment used in previous work to test an incremental clustering algorithm's robustness to data point arrival order [55]. In the experiment, a dataset is ordered in two specific ways:

Round-Robin Randomly determine an ordering of ground-truth clusters. Then, construct a data point arrival order such that the ith data point is a member of cluster $i \texttt{mod} K$, where K is the number of clusters and `mod` returns the remainder when its first argument is divided by its second.

Sorted Randomly determine an ordering of ground-truth clusters. All points of cluster C_i arrive before any point of cluster C_{i+1} arrives.

As in previous work, we perform a robustness experiments with the ALOI dataset. Table 3.4 shows that GRINCH achieves higher dendrogram purity than both PERCH and mini-batch HAC (with 2 different batch sizes) on data ordered using the Round Robin ordering scheme. Under this arrival order, MB-HAC performs poorly showing its lack of robustness. When

Method	Round.	Sort.
GRINCH	0.503	0.457
PERCH	0.446	0.351
MB-HAC (5K)	0.299	0.464
MB-HAC (2K)	0.171	0.451

Table 3.4: DP for adversarial arrival orders (ALOI).

the data is in Sorted order–which makes for easier clustering for MB-HAC–GRINCH outperforms PERCH and is competitive with MB-HAC.

3.5 Related Work

ER is widely studied throughout KB construction and database research. Most common among the techniques of author name disambiguation in particular include hierarchical agglomerative clustering or DBSCAN [29, 66, 51, 52, 105, 48]. However, none of the approaches based on these algorithms are incremental ER and thus they are unsuitable for integrating user feedback and KBs.

The family of online and incremental clustering methods is diverse, however all algorithms in this family optimize for specific linkage functions. PERCH, from which the `rotate` procedure is inspired, performs rearrangments to satisfy a condition similar to complete-linkage [55]. BIRCH is another top-down hierarchical clustering algorithm that attempts to minimize a k-center style cost at each node in the tree [126]. BIRCH also includes a non-greedy reassignment step but has been shown to produce low quality trees in practice. Liberty et al propose a flat clustering algorithm that optimizes k-means cost. Since their algorithm runs in the online setting, after a data point arrives and is assigned to a cluster, it may never be reassigned [69]. While not incremental, some work focuses on designing highly scalable algorithms for specific linkage functions. Particular attention is paid to single-linkage because of its connection to the minimum spanning tree problem. For example, recent work develops massively parallel algorithms for single-linkage [10].

When clustering with linkage functions, probabilistic approaches can provide an alternative to HAC. For example, split-merge Markov Chain Monte Carlo (MCMC) methods perform clustering by randomly splitting and merging clusters according to a proposal function [49]. An algorithm similar to split-merge MCMC has even been used for author coreference [120]. This algorithm employs a custom linkage function on structured records and works by maintaining a forest–each tree corresponding to a cluster–and randomly proposing mergers and splits of various branches. Unlike GRINCH, this algorithm relies on sampling to escape local minima. As the number of items grows, the likelihood of sampling a merge or split that will be accepted decreases rapidly.

Our work is partially inspired by complex linkage functions that are used for clustering. One example is Bayesian hierarchical clustering (BHC)–a recursive, probabilistic, hierarchical model for data [45]. Fitting BHC models is performed by running HAC with BHC as the linkage function. Because HAC is inefficient, randomized approaches for fitting BHC have also been proposed, but each of these methods still runs HAC as a subroutine on small, randomly selected subsets of data [44]. HAC-style algorithms are also used to do probabilistic, hierarchical community detection and alongside learned models for entity resolution [12, 62].

Model-based separation is related to recently proposed definitions of *perfect hierarchical clustering structure* [20, 111], in which pairwise similarities between data points lead to a tree that can be discovered by HAC that has minimal cost. The costs used in these works are variants of Dasgupta's cost [24]. Perfect hierarchical clustering structures are a special case of model-based separation, in which single-, average-, or complete-linkage is used. Model-based separation is strictly more general, allowing for linkage functions that compute the similarity of two point sets arbitrarily, rather than as a function of pairwise data point similarities.

CHAPTER 4

INTEGRATING USER FEEDBACK UNDER IDENTITY UNCERTAINTY IN KNOWLEDGE BASE CONSTRUCTION

4.1 Overview

In the previous chapter we introduced GRINCH, an incremental, hierarchical clustering algorithm, and demonstrated that it is competitive with state-of-the-art approaches for ER. While ER is a pillar of KB construction, ER is insufficient on its own. After ER, KB entity attributes and relationships must be *canonicalized* (Section 2.1.4). During canonicalization, a group of mentions that were deemed coreferent by ER are synthesized to produce a single *inferred entity* that has attributes and participates in relationships with other inferred entities, all evidenced by the underlying mentions. These inferred entities are typically the first-class KB objects with which users interact, and to which users may provide edits.

This chapter opens with a description and formalization of *attribute feedback*: a mechanism by which users provide feedback about the missing and incorrect attributes of inferred KB entities. Unlike attribute feedback, previous work that studies feedback for ER largely focuses on mention-level pairwise constraints, which are insufficient for providing feedback about KB entities, their attributes and relations [113, 68, 108, 75, 76, 127]. For example, using pairwise constraints alone, it is impossible to supply missing attributes and relations or correct noise in the underlying mentions. Similarly, many types of desirable user feedback are inexpressible in the language of pairwise constraints, e.g., specifying a missing attribute, such as a missing email address or affiliation in a KB of scientists (Example 2). Because the number of possible pairwise constraints is large, collecting pairwise feedback introduces an additional challenge of designing specialized strategies for selecting which pairs to label and

in what order [113, 114, 32]. We note that it is also often undesirable for users to interact with a KB at the mention-level.

Next, we describe our approach for integrating attribute feedback into KBs via GRINCH. As previously discussed, one of the challenges is resolving identity uncertainty inherent in the feedback. For concreteness we provide the following example (which is similar in flavor to Example 1 from Chapter 1:

Example 5. *Consider a KB entity,* `Rajarshi Das`[1]*, whose underlying mentions refer to two distinct real-world scientists,* $e_1^\star$ *and* $e_2^\star$*.* `Rajarshi Das` *(the KB entity) is browseable via a profile page. While browsing, a user who is familiar with* $e_1^\star$ *discovers that the profile is missing a link to a homepage. The user supplies attribute feedback that includes the homepage for* $e_1^\star$*; call this attribute feedback* f_1*. Later, while browsing* `Rajarshi Das`*, another user who is familiar with* $e_1^\star$ *but not* $e_2^\star$ *discovers a publication,* P*, written by* $e_2^\star$*. The user provides attribute feedback claiming that* `Rajarshi Das` *did not write* P*; call this feedback* f_2*. Ideally, the KB should split* `Rajarshi Das` *into two KB entities, one corresponding to* $e_1^\star$ *and the other corresponding to* $e_2^\star$*. In doing so, the KB must decide to which of the two profiles* f_1 *should be applied.*

The decision of to which KB entity f_1 applies arises because of identity uncertainty with respect to the target of f_1. In order to address identity uncertainty inherent in the feedback, our approach to integration is founded on the idea that feedback should participate in ER alongside standard mentions. In addition to naturally resolving identity uncertainty, including feedback in ER has additional benefits: the feedback provides evidence for mergers and splits, which makes ER more robust and effective. This is easily seen when contrasting our approach with more static approaches that apply feedback by *overwriting* entity attributes and relations in a post-hoc step, and that do not reconsidering ER decisions.

[1]This example is inspired by ER errors that exist in DBLP as of this writing.

We note that a naïve application of our approach opens opportunities for additional ER errors, e.g., if the user feedback is associated with an incorrect inferred entity, or causes spurious mergers and splits. Specifically, consider the second piece of feedback (f_2) in Example 5. Before the feedback is integrated, it is incompatible with `Rajarshi Das`, since the KB believes the entity to have authored P, which f_2 explicitly negates. Ideally, ER would associate the feedback with `Rajarshi Das`, despite the inconsistency, and use the negated publication attribution as evidence in support of splitting the KB entity's underlying mention cluster.

To enable this, we advocate for the representation of attribute feedback as *feedback mentions* (FMs), which include two components: a *packaging* and a *payload*. The packaging contains attributes used by ER to determine initial placement of the feedback within the GRINCH tree. Afterward, attributes contained in the payload are used to: i) introduce missing attributes, ii) correct spurious attributes, and iii) influence mergers and splits among the underlying mentions. Returning to Example 5, a FM with packaging that includes attributes of R_1 help guide ER toward associating the feedback with the mentions that constitute KB entity `Rajarshi Das`. Afterward, the FM's payload, which includes the negation of P, signal to ER to correctly split the mentions into two inferred entities.

Although attribute feedback is capable of correcting arbitrary KB errors—including noisy and missing data—in this chapter our experiments focus on using the feedback to recover from mistakes in ER; experiments with noisy and missing data appear in subsequent chapters. In this chapter, we present the results of two experiments in the context of *author disambiguation*–a particular instantiation of ER. In the first, we automatically generate user feedback that includes an author's areas of expertise—represented as a set of keywords. In the second experiment, we generate user feedback that identifies missing and incorrectly attributed publications with respect to a set of currently inferred KB entities. We propose 3 baseline approaches for integrating user feedback and 2 feedback simulation schemes, and measure the number of pieces of feedback required to recover the ground-truth

entities under each experimental setting. Our results show that our proposed approach based on FMs outperform the baselines in 70% of experimental conditions. This work initiates the investigation of how to integrate attribute feedback amidst identity uncertainty in KBs, an unexplored and important problem whose solution can dramatically improve the effectiveness of users in the process of KB construction.

4.2 Formal Models of KB Objects and Attribute Feedback

We begin with the notation used henceforth to describe mentions, inferred entities, attributes and relationships. Then, we describe a simple process for canonicalization. Finally, we introduce and formalize attribute feedback, in which users provide feedback with respect to inferred entity attributes and relations.

4.2.1 Notation for KBs and ER

Formally, a KB is comprised of a set of mentions $\mathcal{M} = \{x_0, \cdots, x_n\}$ which refer to a set of ground-truth entities $\mathcal{E}^\star = \{e_0^\star, \cdots, e_k^\star\}$. Each mention, x_i, corresponds to exactly 1 ground-truth entity, denoted $e^\star(x_i)$. The goal in ER is to construct a partition of the mentions, $\hat{\mathcal{E}} = \{\hat{e}_0, \cdots, \hat{e}_l\}$ as similar to $\mathcal{E}^\star$ as possible. Each $\hat{e} \in \hat{\mathcal{E}}$ is known as an *inferred entity*. Thus, in Example 5, `Rajarshi Das` is an inferred entity.

Mentions are comprised of *attributes*, which serve as evidence of inferred entity attributes and relations[2]. Formally, each mention, $x \in \mathcal{M}$, has a corresponding set of attributes, $\mathcal{A}(x) = \{a_0, \cdots, a_m\}$. Any subset of mentions, e, also has a corresponding set of attributes, $\mathcal{A}(e)$, that is derived from its underlying mentions in a process called *canonicalization*. For example, a simple method of canonicalization is one in which the attributes of any set of mentions, e, are derived by forming the union of attributes of over each mention in

[2]For the remainder of the thesis, we group both inferred entity attributes and relations together using the term *attributes*.

the set. This model of mentions, entities and attributes is reminiscent of previous work in ER [101].

4.2.2 Hierarchical Canonicalization

Since our approach is based on GRINCH, we describe a hierarchical process for canonicalizing inferred entities. Recall that GRINCH can be used to build a tree, $\mathcal{T}$, over a KB's underlying mentions. Formally, each leaf, $l \in \mathcal{T}$, stores a unique mention, $l.x = x_i$ and each internal node, $v \in \mathcal{T}$ represents that set of mentions stored at its descendant leaves, $\texttt{lvs}(v)$. Each node $v \in \mathcal{T}$ stores an *attribute map*, $m : A \to \mathbb{R}$, that maps a set of attributes to their corresponding *weights*. The attribute map at each leaf, $l.m$, maps all attributes in $l.x$ to 1[3]. The attribute map of an internal node, $v.m$, is constructed via *canonicalization*. For now, consider a canonicalization procedure that constructs that attribute map of an internal node, v, as follows:

$$v.m[a] = \sum_{c \in \texttt{ch}(v)} c.m[a],$$

where $\texttt{ch}(\cdot)$ returns the children of its arguments and a is an attribute. In words, the weight of an attribute in $v.m$ is the sum of that attribute's weight in v's children's maps. A subset of mentions *exhibits* an attribute a if the weight of a in the corresponding attribute map is greater than 0. Note that a similar style of canonicalization could be applied to a flat clustering of a collection of mentions.

4.2.3 Attribute Feedback

Despite significant study, KB construction approaches are prone to error. Regardless of which stages of the pipeline are noisy, errors produced during KB construction manifest

[3]Mapping attributes to weights allows for modeling the *strength* of various attributes. For example, attributes extracted from a data source that is known to be noisy may have lower weights.

similarly: they yield inferred entities with spurious and/or missing attributes and relationship. Human users are well-situated to detect and correct these errors.

Recognizing that users tend to interact with KBs in an *entity-centric* manner, e.g., through an entity's profile page in a KB like DBLP, we propose *attribute feedback*, a style of user feedback that is contributed at the entity-level. Generally, an attribute feedback is a statement about a ground-truth entity that identifies some of that entity's attributes and optionally identifies some attributes that the entity *does not* exhibit—called *negations*. For example, the statement, "`Fernando Pereira`, *who is affiliated with the organization Google, was never affiliated with the Instituto de Telecomunicações in Portugal*" is attribute feedback that includes a negation. Formally, we define an attribute feedback as follows:

Definition 4 (Attribute Feedback)**.** *Let* $\mathbb{A}$ *be the set of all attributes and let* $\overline{\mathcal{A}}(e^\star) = \mathbb{A} \backslash \mathcal{A}(e^\star)$*. An* ***attribute feedback*** *is a tuple* (p, n) *with an unknown, target ground-truth entity,* $e^\star \in \mathcal{E}^\star$*, where* $p \subseteq \mathcal{A}(e^\star)$*,* $p \neq \emptyset$ *and* $n \subseteq \overline{\mathcal{A}}(e^\star)$*.*

In words, a user providing attribute feedback has an intended ground-truth entity in mind when creating the feedback. The feedback includes a subset of the attributes of that ground-truth entity as well as a subset (potentially empty) of the attributes that the ground-truth entity does not exhibit. A more in depth treatment of attribute feedback can be found in our previous work [56].

4.3 Feedback Mentions

Assuming that information extraction is performed by an external process, at this point in the discussion, the tools we have developed can be used for KB construction. In particular, GRINCH is used to construct a hierarchical clustering over an initial collection of mentions. Canonicalization is performed bottom-up in the tree (Section 4.2.2). Afterward, each node exhibits all attributes in its attribute map that have strictly positive weight. A tree-consistent partition is extracted from the tree, where each cluster of the partition is an

inferred entity. When new mentions arrive, they are incrementally added to the tree by GRINCH, canonicalization is rerun, and a new partition is extracted.

Now, we describe how user feedback is handled. Given a unit of feedback, a naïve integration approach is to determine which node in the tree is the intended target of feedback, and modify its corresponding attribute map accordingly. However, since new data (including user feedback) can cause GRINCH to reorganize the tree, internal nodes are subject to deletion, e.g., if one of its children is moved during the `graft` subroutine (Algorithm 1). If a node modified by user feedback is slated for removal, the KB would need knowledge that feedback had been applied to that node as well as additional logic to determine the new target of the feedback (i.e., resolve identity uncertainty); otherwise, the feedback must simply be removed.

As previously discussed, we propose an alternative approach founded on treating user feedback *as mentions*, which we call *feedback mentions* (FMs). Just like mentions, each unit of feedback possesses attributes and is housed in a leaf node in the tree. The addition of feedback to the tree may precipitate the splitting of an inferred entity or spur the `graft` subroutine and cause a merger. Since feedback is stored in the leaves of the tree, it is never in danger of being removed (as it is in the naïve integration strategy above). Identity uncertainty is naturally handled by GRINCH, which is free to reorganize mentions and feedback as necessary.

However, allowing GRINCH to resolve identity uncertainty may result in additional errors. As mentioned in the overview of this chapter, a likely source of such errors are negations included in attribute feedback (Section 4.2.3). By definition, negations are at odds with the very inferred entities they target, and could cause unnecessary splits or be placed incorrectly in the tree via GRINCH's initial nearest neighbor search. To address this concern, we propose and study two types of FMs. The first contains a single attribute map, called *packaging*, similar to standard nodes in the hierarchy. The second representation possesses a second attribute map called *payload*, which is designed to obscure certain attributes until

after the initial placement of the feedback. Before we describe both types of FMs, we briefly discuss attribute feedback that contain negations.

4.3.1 Negations

One of the primary functions of user feedback is to identity spurious attributes and relations. For example, in the scientific KB example above (Example 5), the user supplies feedback claiming that `Rajarshi Das` *did not* author publication P This negation is represented by an attribute with a corresponding *negative weight*. We assume that the linkage function used by GRINCH will produce a low score for two nodes, one which contains an attribute that the other negates; or, more technically, two nodes that possess differently signed weights for the same attribute. Low linkage scores deter nodes from residing close to one another in the tree. Assuming that a tree-consistent partition of the GRINCH tree is selected via a threshold, attribute feedback that contains negations can also initiate splits of an inferred entity into multiple by decreasing the linkage scores at certain nodes below the threshold.

In addition to splitting inferred entities, negatively weighted attributes can also be used to effectively *remove* incorrect attributes from their ancestors in $\mathcal{T}$. Consider two nodes, v and v', such that $v.m[a] = 1$ and $v'.m[a] = -1$. Despite the negated attribute, assume that there exists sufficient evidence to make v and v' siblings. Then, once their parent, p, is canonicalized (Section 4.2.2), $p.m[a] = 0$, effectively removing the attribute a from p.

4.3.1.1 Packaging and Payload FMs

Now we turn to the two types of FMs. The first type we call *Packaging Only* (Pack FM). This style of FM contains a single attribute map. The primary difference between Pack FMs and standard mentions is that mentions implicitly map all of their attributes to 1 and Pack FMs may map their attributes to any real-values.

The second type of FM (which we advocate and simply call FM) is composed of two attribute maps, rather than one. The first is called the *packaging*, $f.m_{\texttt{pack}}$, and second is

called the *payload*, $f.m_{\texttt{pay}}$. The packaging contains a set of attributes used to guide the initial placement of the feedback among the KB mentions and entities. Precisely, in the first step of GRINCH, a nearest neighbor search is performed in order to find the new data point's initial placement in the tree; the corresponding linkage scores between mention pairs would only be computed using the mentions' packagings. The second set of attributes is used during the initial canonicalization of the parent of the feedback, and also involves the attributes in the payload. Specifically, for a parent p:

$$p.m_{\texttt{pack}}[a] = \sum_{c \in \texttt{ch}(v)} c.m_{\texttt{pack}}[a] + c.m_{\texttt{pay}}[a].$$

Internal (i.e., non leaf) nodes always contain empty payloads.

As a concrete example, consider an inferred entity that exhibits an incorrect attribute, $\bar{a}$, that is derived from a noisy mention and not from ER. A user can create corresponding attribute feedback negating $\bar{a}$, which yields a choice of whether to negation should appear in the packaging or the payload (with a weight of -1). Assuming the linkage function assigns a low score to two nodes with the same attribute but differently signed corresponding weights, were the negation to appear the packaging, the feedback may not be compatible with its intended target during GRINCH's initial nearest neighbor search. The incompatibility between the feedback and its intended target may be avoided by storing the negation in the FM's payload. In such a case, the negation would be applied during canonicalization and might result in the intended outcome: the removal of $\bar{a}$ from the inferred entity.

4.4 Experiments

In order to empirically study the two styles of FMs, we conduct experiments in author coreference. We use the Rexa author disambiguation dataset, which includes 8 author *canopies*; each canopy contains ambiguous mentions of authors with the same first initial and last name [21]. The mentions are derived from publications and contain: coauthors,

titles, publishing venue and year of publication. The goal is to partition the mentions by real-world author. We measure the number of units of feedback supplied until the ground-truth partition of the mentions is discovered. For added control over the experimental setting, feedback are generated synthetically according to a prescribed process. Integrating user feedback with KBs under identity uncertainty has not been the subject of significant study. Therefore, we propose a new experimental framework, which is detailed below.

4.4.1 Representations Compared

We propose and compare the following baseline methods:

1. **Packaging and Payload (FM)** - the approach advocated in this work. Feedback is constructed with packaging and payload attribute maps.

2. **Pack FMs (pack)** - similar to **FM**s, but without payloads. All attributes that would have been included in a payload are instead added to the corresponding packaging.

3. **Hard Assignment (assign)** - generate feedback with both packaging and payload as in **FM**s. But upon insertion, find the node $v \in \mathcal{T}$ to which it would have been made a sibling by GRINCH and *permanently* assign the feedback to v. If mentions are ever removed from v (e.g., by a `graft`), remove and delete all feedback assigned to v, since the target of the feedback is now unknown (due to identity uncertainty in the feedback).

4. **Hard Mention Assignment (assign-m)** - similar to the **assign** approach but the feedback must be assigned to a mention in $\mathcal{T}$. Since mentions are atomic (rather than ephemeral, like inferred entities), the assigned feedback is never deleted. However, if the mention is placed incorrectly at first, it requires additional feedback to resolve the error.

4.4.2 Threshold-aware GRINCH

We make small modifications to GRINCH to account for the fact that we now care about extracting a tree-consistent partition from the hierarchy it builds, rather than maximizing dendrogram purity (as in Chapter 3). Recall that GRINCH makes use of a linkage function, g, that scores the compatibility of any two nodes in the tree. Each node, v, stores its *linkage score*, $v.\sigma$, where the linkage score of a node is computed by evaluating g on the attribute maps (i.e., packaging) of its two children, $\texttt{ch}(v)$. The linkage score of each leaf is positive infinity. Once the linkage score of all nodes in a tree, $\mathcal{T}$, are computed, the set of inferred entities, $\hat{\mathcal{E}}$, can be extracted from $\mathcal{T}$ using a threshold, τ (a hyperparameter). In particular, the inferred entities correspond to the tallest nodes in $\mathcal{T}$ whose descendants all possess linkage scores greater than or equal to the threshold. At all times, we use a threshold τ, to identify the set of inferred entities.

Given these modifications, we make the following modifications to the `rotate` and `graft` subroutines:

4.4.2.0.1 `rotate`. Now, no rotations are performed at the ancestors of inferred entities since these rotations have no tangible effect.

4.4.2.0.2 `graft`. A `graft` invoked from a node whose linkage score is below the threshold terminates immediately and no further grafts are attempted. If a `graft` is initiated from p, and $p.\sigma > \tau$, search the leaves of $\mathcal{T}$ for, v', the highest scoring leaf with p that is not a descendant of p. Test whether

$$g(p, v') > \max\{g(p, \texttt{sib}(p)), g(v', \texttt{sib}(v')), \tau\}$$

i.e., p and v' are scores higher together than with their respective siblings, and also that their linkage score is higher than the threshold τ. If the test succeeds, make v' the sibling of p, re-invoke the `graft` subroutine from $\texttt{par}(p)$. If the test fails, consider 3 cases:

1. if $g(p, v') \leq \tau$ then re-invoke the `graft` subroutine from $\texttt{par}(p)$;

2. if $g(p, \texttt{sib}(p)) > g(p, v')$ then repeat the test between $\texttt{par}(p)$ and v';

3. if $g(v', \texttt{sib}(v')) > g(p, v')$, then repeat the test between p and $\texttt{par}(v')$.

As before (Chapter 3), the `graft` subroutine recursively attempts mergers between ancestors of p and nodes compatible with those ancestors in $\mathcal{T}$. But now, a merger between two nodes in $\mathcal{T}$ can only occur if: 1) both nodes score higher with each other than with their siblings and 2) their resultant parent has a linkage score higher than the threshold, i.e., the two nodes belong to the same inferred entity according to the threshold.

4.4.3 Simulating Feedback

We simulate positive and negative feedback using node *purity* and *completeness*. A node $v \in \mathcal{T}$ is *pure* if, for some i:

$$\forall l \in \texttt{lvs}(v), \quad e^\star(l) = e_i^\star,$$

i.e., all of mentions stored at the leaves of v correspond to the same ground-truth entity, $e_i^\star$. A node $v \in \mathcal{T}$ is *complete* if, for some i:

$$\{l \in \texttt{lvs}(\mathcal{T}) : e^\star(l) = e_i^\star\} = \{l' \in \texttt{lvs}(n) : e^\star(l') = e_i^\star\},$$

i.e., that v's leaves contain *all* mentions $e_i^\star$.

To generate both positive and negative feedback, we sample an intended *destination* and an intended *target*. The destination is a particular node in the tree to which the feedback is intended to be similar. The target of the feedback is a different node that the feedback is intended to be merged with, or that the feedback intends to separate from the target's current inferred entity. Note that even with full knowledge of the destination and the target, it would require significant reasoning to design feedback that would incite a specific tree

rearrangement. At a high level, this is because, the nodes in $\mathcal{T}$ exhibit complex relationships with each other through GRINCH. To give a concrete example, even if a FM were design to have a specific nearest neighbor, we must also reason about `grafts` that would occur after the initial placement of the FM and the subsequent canonicalization.

4.4.3.1 DETAILED and CONCISE Positive Feedback

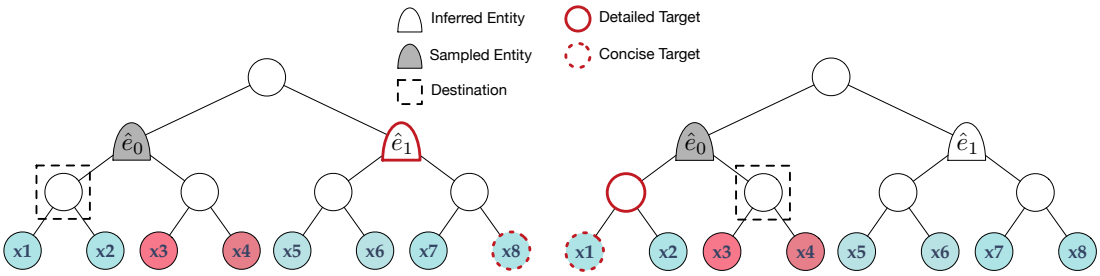
Positive feedback is constructed with the intention of merging two subentities via a `graft`. To generate positive feedback, select the root of a pure and incomplete inferred entity, r, to be the destination of the feedback. Then, randomly select a mention, x, that is of the same ground-truth entity as the leaves of r, but is not a descendant of r. If constructing CONCISE feedback, x is the target of the feedback; if constructing DETAILED feedback, traverse the ancestors of x until s, the first ancestor of x whose parent is impure. The node s becomes the target of the feedback. See Figure 4.1a for a visual illustration.

4.4.3.2 DETAILED and CONCISE Negative Feedback

Negative feedback is constructed with the intention of splitting an inferred entity. We simulate negative feedback by randomly sampling an impure inferred entity and finding its root, r'. We construct the destination of the feedback by randomly sampling a mention $x' \in \texttt{lvs}(r')$ and finding s', the ancestor of x' closest to the root of $\mathcal{T}$ that is pure. If constructing CONCISE feedback, sample a mention $x'' \in \texttt{lvs}(r') \setminus \texttt{lvs}(s')$ to be the target; if constructing DETAILED feedback, traverse the ancestors of x'' until s'', the ancestor of x'' closest to the root of $\mathcal{T}$ that is pure. In both cases, s'' becomes the target of the feedback. See Figure 4.1b for a visual illustration.

4.4.4 Setup

Our experimental setup is composed of two phases. In the first phase, we use GRINCH to build a clustering, $\mathcal{T}$, of the initial set of mentions. The second phase proceeds in rounds. At the start of round t, a set of inferred entities, $\hat{\mathcal{E}}_t$, is constructed using a threshold, τ,



(a) Positive feedback generation.

(b) Negative feedback generation.

Figure 4.1: **DETAILED and CONCISE feedback.** To generate either positive or negative feedback, begin by randomly sampling an inferred entity. Then, sample a *destination*–the root of a pure subtree that is also a descendant of the sampled entity. The packaging of the feedback contains the attributes at the destination. Finally, sample a *target*, which is used to construct the feedback's payload. The target is a sampled mention in the CONCISE setting, or the largest, pure ancestor of a sampled mention in the DETAILED setting.

tuned at training time on a development set (see [57] for more details). If $\hat{\mathcal{E}}_t = \mathcal{E}^\star$, then the episode terminates. Otherwise, we simulate user interaction by generating feedback, f_t, made with respect to a randomly selected inferred entity, $\hat{e} \in \hat{\mathcal{E}}_t$. Then, f_t is added to $\mathcal{T}$ using GRINCH, potentially triggering a repartitioning of the mentions. No more than 400 rounds are permitted[4]. Although rare, if after 400 rounds, the ground-truth entities have not been discovered (i.e., $\hat{\mathcal{E}}_t \neq \mathcal{E}^\star$), the method is recorded as having taken $400 + d$ rounds, where d is the number of mentions that would need to be swapped to discover $\mathcal{E}^\star$. We measure the mean number of rounds required to discover $\mathcal{E}^\star$ for each method, repeated over 25 trials, and report a paired-t statistic (and corresponding significance level) with respect to FMs and each other competing representation.

4.4.5 User Feedback about Author Expertise

Our first experiment resembles a scenario in which users interact with a KB of scientists and provide feedback with respect to the KB's belief about a scientist's expertise. The

[4]During experimentation, we find that if convergence to the ground-truth partition is not achieved after 400 rounds, convergence is unlikely to occur.

expertise of an inferred entity is represented as a bag of key phrases drawn from the titles of its underlying mentions. Users supply missing keywords and identify incorrect keywords. In this experiment, packaging contains the set of attributes at the sampled destination and the payload contains the keywords at the target (generated from mention titles).

Table 4.1a contains the results of the paired t-test between each baseline method and our proposed approach (FM), under DETAILED and CONCISE feedback generation schemes, with respect to the number of pieces of feedback required to discover the ground-truth partition of the mentions. Each row in the table represents a canopy in which the experiment is performed, and each column corresponds to a baseline method and feedback generation setting. Each cell contains the difference between the mean number of rounds required by the FM approach and a baseline approach to discover the ground-truth partition (higher is better). Positive numbers are bolded; asterisks (*) indicate statistical significance ($p < 0.05$) and two asterisks (**) indicate statistical significance ($p < 0.01$).

4.4.6 Authorship Feedback

Our second experiment resembles the scenario in which a user browses a KB of scientist profiles, similar to Google Scholar[5], and identifies incorrectly assigned and missing publications. Similar to the first experiment, the packaging is copied from the sampled destination. However, here, payloads contain titles stored in the sampled targets. Table 4.1b contains the results for this experiment of the paired t-test in the aforementioned format.

4.4.7 Discussion

The paired-t statistic results compares our proposed approach (**FM**) to the three baseline approaches. We find that **FM** outperforms pack in both the DETAILED and CONCISE settings of Experiment I on all but two of the canopies. In seven of the fourteen canopies, FM outperforms **pack** in a statistically significant way. We hypothesize that these results

[5]https://scholar.google.com/

indicate the importance of separating the packaging and payload attributes. We hypothesize that **pack** suffers from placing edits in incorrect locations in the tree due to combining the two. In Experiment II, we find similar results in the CONCISE setting, but find that **pack** requires fewer feedback rounds compared to FM in the DETAILED setting (not statistically significant). We hypothesize that pack's improved performance in the DETAILED setting could be due to negative title attribute feedback helping in correctly placing mentions in the tree by diverting the feedback mention away from all mentions with the negative title attributes. In comparing, **FM** and **assign** we find that our proposed approach typically performs better in Experiment II while the baseline performs better in Experiment I. We note that the feedback in Experiment I is more ambiguous than Experiment II. We hypothesize that **assign**'s performance in Experiment I is due to deleting feedback that was placed incorrectly in the tree. We find that FM frequently outperforms **assign-m** in both Experiment I and II. We hypothesize that **assign-m** frequently attaches feedback to the wrong mention. This highlights the dependence on the learned linkage function for accurately integrating feedback.

4.5 Related Work

Effective utilization of user feedback has been the subject of significant study in the context of KB construction. Early work, like NELL, primarily enlists humans for labeling data, which are used to train downstream models [15]. Other work has used active learning in training relation extraction models [4]. Another approach employed by the DeepDive system asks humans to identify relevant features by writing feature extraction rules in support of KB construction [92, 3]. More recently, the Snorkel system asks users to write labeling heuristics, which are combined automatically and used to label dataset in support of model training [90].

The study of leveraging user feedback in ER has primarily focused on the solicitation of pairwise feedback. For example, given a set of mention pairs, the CrowdER system

automatically prunes the set of pairs that are highly unlikely to be coreferent, and then constructs crowdsourcing HITs to collect binary labels for the remaining pairs [113]. In other similar work, human are asked to identify matching mentions across databases in data integration [68]. Recent work studies online ER with an oracle, in which the goal is to design efficient strategies for soliciting humans for pairwise constraints among mentions [108, 75, 76].

Recent work in author coreference also involves humans-in-the-loop [127]. This work discusses both pairwise constraints as well as *identity constraints*. Unlike our work, their identity-level feedback is treated as a collection of pairwise constraints. As we point out, feedback that can be reduced to a set of pairwise constraints is insufficient for general KB feedback as pairwise feedback is only designed for correcting errors in ER (and not general KB errors). Similarly, many examples of user feedback are inexpressible using pairwise constraints.

The most closely related work to this thesis is our preliminary study of incorporating user feedback in the context of data integration [118, 119]. In this work, users supply pairs of mention-like records that posses either should-link or should-not-link factors, which either softly repel the pair or encourages their merger.

	Detailed			Concise		
Canopy	vs. **assign**	vs. **assign-m**	vs. **pack**	vs. **assign**	vs. **assign-m**	vs. **pack**
allen_d	-2.07^{*}	**4.03****	**3.41****	**1.88**	**2.20***	**0.75**
blum_a	-1.18	**1.59**	**1.27**	-0.85	**0.20**	**0.61**
jones_s	**1.37**	**7.74****	**6.67****	**1.11**	**0.65**	**4.04****
lee_l	-1.53	**1.02**	**4.04****	-0.22	-0.28	**0.95**
moore_a	**0.90**	**3.25****	**3.66****	-0.30	-0.71	-0.49
robinson_h	-0.99	-3.60^{**}	-0.81	-0.76	-0.61	**0.27**
young_s	-0.71	**1.09**	**4.87****	**0.76**	**0.45**	**2.17***

(a) Experiment I: keyword feedback.

	Detailed			Concise		
Canopy	vs. **assign**	vs. **assign-m**	vs. **pack**	vs. **assign**	vs. **assign-m**	vs. **pack**
allen_d	**4.82****	**2.67***	-1.96	**5.86****	**2.01**	**1.98**
blum_a	**0.11**	-0.66	-0.98	**0.98**	**1.03**	**5.70****
jones_s	**1.49**	**1.99**	**0.85**	**1.00**	**1.02**	**3.03****
lee_l	**0.64**	**1.06**	-1.03	-1.44	**0.47**	-0.58
moore_a	**1.30**	**1.97**	-0.90	**1.94**	**1.09**	**1.05**
young_s	-1.87	**0.76**	-1.88	**0.93**	**0.91**	**0.97**

(b) Experiment II: title feedback.

Table 4.1: **Paired-t statistic.** Each cell represents that difference in mean number of feedback-rounds required to discover the ground-truth entities over 25 runs between a baseline, denoted by the column heading, and our proposed approach (FM). Positive numbers indicate that FM requires *fewer* rounds of feedback than its competitor (larger numbers are better). Two asterisks (**) indicates that the statistic is significant at a 0.01 significance level; one asterisk indicates statistical significance at the 0.05 level. The `mcguire_j` canopy is excluded from Tables 4.1a and 4.1b and the `robinson_h` canopy is excluded from Table 4.1b since in these canopies, either: 0 or 1 edits are required to discover the ground-truth entities across baselines.

CHAPTER 5

EFFICIENT REASONING ABOUT SOURCES OF ERROR AND IDENTITY UNCERTAINTY DURING FEEDBACK INTEGRATION

The previous chapter explores the use of GRINCH as a feedback integration algorithm for KBs that exhibit identity uncertainty. A key ingredient in the proposed strategy is the representation of feedback as mentions with packaging and payload. To evaluate our proposed approach, we used real author coreference data with simulated user feedback. While, in theory, our framework can correct errors from imperfect ER and from noisy data, the experiments discussed in the previous chapter only test recovery from ER errors.

In this chapter, we study our proposed approach in the context of `OpenReview.net` (OpenReview)—a real KB with real user feedback. As a large software project, fitting our feedback integration scheme to OpenReview requires adherence to additional constraints. Among them is a demand to a reduction in the number of additional database objects GRINCH creates (i.e., tree structure). This leads to our first technical contribution of this chapter: the XGRINCH algorithm, a variant of GRINCH with arbitrary branching factor. By virtue of *collapsing* some internal structure, XGRINCH maintains trees of significantly smaller size than GRINCH. The decision of whether to collapse particular internal structure is made using a tree-building invariant, which we introduce in Section 5.1. Furthermore, since the feedback integration framework is responsible for partitioning the underlying data at all times, we also present GRINCH-SHALLOW, which stores a *forest* of trees—each tree representing an inferred entity (i.e., a cluster of mentions)—rather than a monolithic binary tree. Like XGRINCH, GRINCH-SHALLOW also reduces the number of internal nodes required. But, by eliminating internal structure above the partition, GRINCH-SHALLOW also admits a *short-circuiting* of the the `graft` operation—a central and expensive subprocedure

of GRINCH. We show that XGRINCH and GRINCH-SHALLOW may be combined to produce XGRINCH-SHALLOW (XGS), which reduces the number of internal nodes required by more than 60% (in comparison to GRINCH), and which is the most efficient algorithm in the GRINCH family.

Empirically, we find that working with real user feedback also carries novel challenges. Unlike the previous chapter, many of the mentions in OpenReview are noisy, and much of the corresponding user feedback is aimed and correcting their downstream effects. This exposes the problem of *indeterminate error sources*, i.e., uncertainty with respect to whether a particular error is caused by faulty ER or corrupt mentions. We show that the precise *location* of feedback in a tree constructed by GRINCH (or XGS) implicitly determines whether the feedback is treated as a correction to ER, noisy data, or missing data. This observation leads to the realization that, for a certain classes of models, GRINCH is brittle when it comes to recovering from errors that stem from noise in the data. In part, this brittleness is due to the arbitrarily deep, binary trees, built by GRINCH. We explain theoretically and demonstrate empirically that the combination of XGS and the use of packaging and payload can partially remedy this brittleness.

5.1 XGRINCH

In this section we present XGRINCH, a variant of GRINCH that builds trees with arbitrary branching factor, inspired by a requirement to maintain smaller trees. At a high-level, XGRINCH *collapses* (i.e., flattens) a binary subtree when the linkage score of each internal node is equally "good" under the model. We begin our discussion with the invariant that is central to the design of XGRINCH. Then, we describe the modifications of GRINCH necessary to support the new algorithm.

5.1.1 Non-binary Subtree Invariant

In its original presentation GRINCH builds full, binary trees, exclusively. By virtue of being full and binary, these trees always have $2n - 1$ total nodes, where n is the number of underlying mentions. We propose XGRINCH, which reduces the number of internal nodes by building trees with arbitrary branching factor.

More formally, let $\mathcal{T}_B$ be a full binary tree and let $v.\sigma$ be the linkage score of internal node $v_i \in \mathcal{T}_B$. Recall that the linkage score of v represents the affinity of v_i's children for one another. Then, in a tree, $\mathcal{T}$, built by XGRINCH, the following invariant is maintained:

Invariant 1. *For any group of siblings, $S = \{v_i, \ldots, v_j\}$ there exists a binary tree over S for which each internal node has the same linkage score.*

In other words, if a binary tree over S can be built such that each internal node has the same linkage score, XGRINCH may collapse that tree, making all its leaves siblings of one another.

Note that Invariant 1 *may* collapse a subtree that has only one construction where all internal nodes have the same score. We argue that collapsing such subtrees is also desirable in the context of general clustering. This is because: i) the operation condenses the tree (smaller memory requirements are better), and ii) it is unlikely to effect the dendrogram purity of the overall tree. To see why, note that since all internal nodes have the same score, either: all nodes are likely members of the same cluster, or all nodes are likely members of distinct clusters.

5.1.2 XGRINCH Subroutines

Maintaining Invariant 1 requires that a number of modifications be made to the GRINCH algorithm. For clarity of discussion, let each node $v \in \mathcal{T}$ maintain a score, $v.\sigma$, which represents the affinity of its children for one another. Thus, a node v with score $v.\sigma$ and multiple children indicates that those children can be arranged into a binary tree in which each internal node has score $v.\sigma$. As before, each leaf in $\mathcal{T}$ has score $+\infty$.

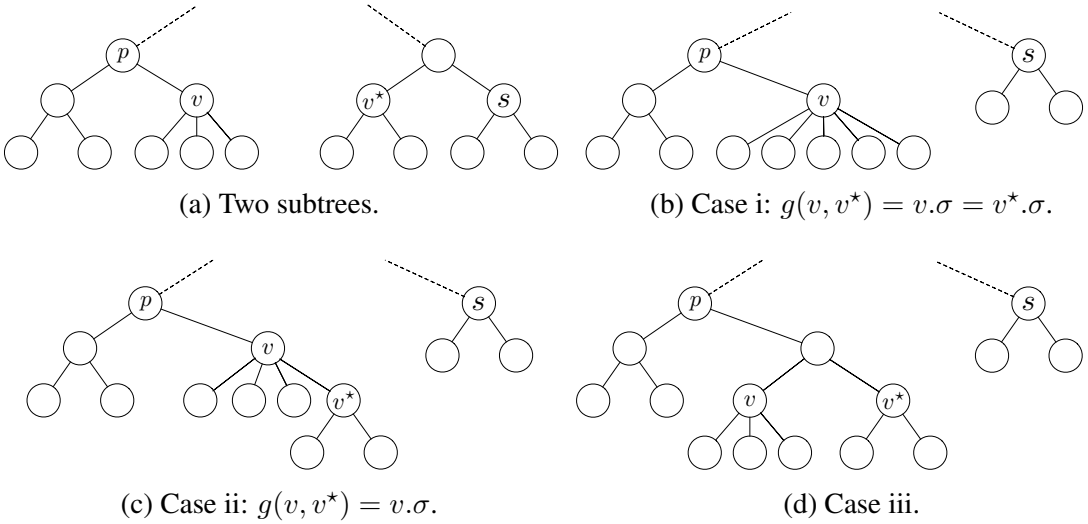
Like the original algorithm, when a new point is inserted, XGRINCH initially adds it as a sibling of its nearest neighbor. Then, rotations are applied, followed by grafting and restructuring. Each of these subroutines requires modification in order to maintain Invariant 1. As before, after tree modifications, ancestor nodes must be updated. However, the `xupdate` subroutine in XGRINCH also serves to maintain Invariant 1. These modifications are discussed below.

5.1.2.1 `xrotate`

The `xrotate` subroutine is applied after a node x_i is inserted next to its nearest neighbor l. The `xrotate` subroutine checks if the score between x_i and l is greater than or equal to l and its current siblings, i.e, $f(x_i, l) \geq \texttt{par}(l).\sigma$. Like in `rotate`, if $f(x_i, l) > \texttt{par}(l).\sigma)$, then x_i and l are made siblings under a new parent. However, if $f(x_i, l) = l.\sigma$, then x_i is made an *additional* sibling of l. Since all leaves have score $+\infty$, x_i is never added as a child of a leaf, but after at least 1 rotation, this condition may trigger. If neither of these conditions hold, the same checks are repeated at the parent of l (and proceeds until x_i is added to the tree).

5.1.2.2 `xgraft`

After rotations, the `xgraft` operation is attempted. Like the GRINCH `graft` operation, an `xgraft` is invoked from v (an internal node) and performs a *nearest neighbor search* among its non-descendent leaves for $v^\star$—the node to which v has highest linkage score. Afterward, a check is performed that compares: the linkage score at the parent of v, the linkage score between v and $v^\star$, and the linkage score at the parent of $v^\star$. Note that the linkage score at the parent of a node (e.g., v), represents the affinity of that node for its siblings. In GRINCH, if the linkage score between v and $v^\star$ is greater than the linkage scores between v and its siblings as well as $v^\star$ and it siblings, a `graft` makes $v^\star$ the (only) sibling of v, which results in the creation of additional internal structure. But in XGRINCH, an `xgraft` is more nuanced because of XGRINCH's ability to create arbitrary branching factor.



(a) Two subtrees.

(b) Case i: $g(v, v^\star) = v.\sigma = v^\star.\sigma$.

(c) Case ii: $g(v, v^\star) = v.\sigma$.

(d) Case iii.

Figure 5.1: **xgraft.** Starting from an initial tree (Figure 5.1a), the `xgraft` is initiated from v and finds $v^\star$, its nearest neighbor according to the model g. One of 3 distinct tree rearrangements are is performed (Figures 5.1b-5.1d), based on the relationships between: $g(v, v^\star)$, $v.\sigma$, and $v^\star.\sigma$.

In an `xgraft`, first check if the linkage score between v and $v^\star$ is greater than the linkage score at the parent of v and the linkage score at the parent of $v^\star$ (i.e., v's affinity for its siblings and $v^\star$'s affinity for its siblings, respectively). If so, there are 3 cases:

i) If $v.\sigma$, $v^\star.\sigma$ and the affinity between v and $v^\star$ are all equal, then make all children of $v^\star$ siblings of the children of v.

ii) If the affinity between v and $v^\star$ is equal to $v.\sigma$, then make $v^\star$ and child of v.

iii) Otherwise, make $v^\star$ the *only* sibling of v, as in GRINCH.

See Figure 5.1 for a visual illustration of the above cases. It is easy to see that in each case, Invariant 1 is preserved at the site of the graft.

5.1.2.3 `xrestruct`

In GRINCH, after a node, $v^\star$, is successfully grafted, the `restruct` operation is invoked. This operation reorganizes the tree between $z = \texttt{sib}(v^\star)$, the original sibling of $v^\star$ (before

Algorithm 4 $\texttt{xrestruct}(z, r, f)$

while $z \,!=\, r$ **do**
  $A = \bigcup_{a \in \texttt{ancs}(z)} \texttt{sib}(a)$
  $s^\star = \text{argmax}_{s \in A} f(z, s)$
  if $f(v, s^\star) == v.\sigma$ **then**
    $\texttt{makechild}(v, s^\star)$
  else
    $\texttt{makesib}(v, s^\star)$
  $v = \texttt{par}(s^\star)$

the `graft`), and $r = \texttt{lca}(z, v)$, the least common ancestor of z and v. Reorganization of the tree is necessary to maintain GRINCH's theoretical guarantees (Appendix A.1).

In XGRINCH, a similar subroutine is again necessary. The `xrestruct` operation (Algorithm 4) in XGRINCH begins by collecting all ancestors of z (until r) and inspecting their siblings. Let $s^\star$ be the sibling with highest linkage score for z. If the score between z and $s^\star$ is equal to the score among the children of z, make $s^\star$ an additional child of z. Otherwise, make $s^\star$ the (only) sibling of z. After moving $s^\star$, repeat the process from $\texttt{par}(s^\star)$, the parent of $s^\star$ (which may be z). Terminate the `xrestruct` when $\texttt{par}(s^\star) = r$[1]. Again, it is easy to see that after an `xrestruct`, Invariant 1 is maintained. Note that unlike the `restruct` of GRINCH, in XGRINCH the `xrestruct` does not make use of the `swap_l` subroutine. See Figure 5.2 for a visual illustration of `xrestruct`.

5.1.2.4 `xupdate`

After invoking a subroutine that modifies the tree structure (`xrotate`, `xgraft`, or `xrestruct`), a subset of the nodes in the tree will have their internal representations (i.e., features) updated. `xupdate` (Algorithm 5) is a non-trivial operation in XGRINCH since updating features (without additional tree rearrangments) may break Invariant 1. As an

[1]It is possible for r to be deleted during this process if all its children are moved elsewhere. If this is the case, terminate when $\texttt{par}(s^\star)$ is the last remaining descendant of r that was also an ancestor of z

Algorithm 5 `xupdate` (v, r, f)

while $v \,!= r$ **do**
  $s^\star = \text{argmax}_{s \in \texttt{sib}(v)} f(v, s)$
  if $f(v, s^\star) == v.\sigma$ **then**
    $\texttt{makechild}(v, s^\star)$
  else
    $\texttt{makesib}(v, s^\star)$
  $v = \texttt{par}(s^\star)$

example, consider the insertion of a new node x. The features of x are propagated from x to r, the root of the tree, i.e., the path $x \rightsquigarrow r$. Let $s \in x \rightsquigarrow r$ be a node on the path from x to r. Then, after the insertion of x, the features of s are updated to reflect the existence of its new descendent x. Let $|\texttt{sib}(s)| > 1$, i.e., s has more than 1 sibling. Before x arrived, by Invariant 1, there must have existed a binary tree over $S \cup s$ in which all internal nodes had the same score. Since the features of s have changed as a result of the insertion of x, this may no longer be the case. During an `xupdate`, this must be detected and remedied.

An `xupdate` is invoked from a node v. To begin, search $S = \texttt{sib}(v)$, i.e., the siblings of v, for the node, $s^\star$, to which v has highest linkage score. If the linkage score between $s^\star$ and v is equal to the linkage score among v's children, then make $s^\star$ a child of v. Otherwise, make $s^\star$ and v (binary) siblings. In either case, repeat the process from the parent of $s^\star$. See Figure 5.3 for a visual illustration.

5.1.3 Computational Complexity

In the following, let h be the maximum height of a tree, $\mathcal{T}$, and let b be the maximum branching factor (i.e., the maximum number of siblings at any level).

5.1.3.1 `xrotate`

The `xrotate` subprocedure starts from a leaf, l, and computes the model score between a new point, x_i, and the ancestors of l (with the potential to stop early). In the worst

case, `xrotate` may compute the model score between x_i and each ancestor of l yielding computational complexity of $\mathcal{O}(h)$.

5.1.3.2 `xupdate`

Beginning from a node, v, `xupdate` computes the score between v and each of its siblings, incurring a cost of b. The best of its siblings, $s^\star$, is identified and made a sibling of child of v. Afterward, the process repeats, incurring a cost of $b - 1$. Therefore, at each ancestor of v, the cost of an `xupdate` is $\mathcal{O}(b^2)$ for a total cost of $\mathcal{O}(b^2h)$.

5.1.3.3 `xrestruct`

The `xrestruct` is similar to the `xupdate` subroutine except that in `xrestruct` the nodes that are considered (to be moved) is a larger set. In particular, all siblings of the ancestors of v are considered. Thus, each time the siblings of v's ancestors are collected, a cost of $\mathcal{O}(bh)$ model computations is incurred. Afterward, the best sibling is made either a sibling or child of v and the processes is repeated, for a total of $\mathcal{O}(bh)$ repetitions. In total, this incurs a cost of $\mathcal{O}(b^2h^2)$.

5.1.3.4 `xgraft`

An `xgraft` is invoked from a node v at which point a nearest neighbor search is performed. Once $v^\star$, the nearest neighbor of v, is discovered, in the worst case (computationally) a graft is performed. Making $v^\star$ a sibling or child of v is a constant time operation, but afterward, `xrestruct` and `xupdate` are performed. Without additional assumptions, the number of times a graft may be successful is, $\mathcal{O}(n)$ (the total number of points), when initiated, the a single new point is made a sibling or child of the point from which the `xgraft` is initiated. Assuming that nearest neighbor search costs $\mathcal{O}(n \log n)$, this yields a worst case complexity of $\mathcal{O}(n^2 \log n \cdot nb^2h^2)$.

5.1.4 GRINCH-SHALLOW and XGS

GRINCH-SHALLOW represents a modification to GRINCH, which can be applied when the desired result is a flat clustering rather than a full dendrogram. The algorithm is similar in flavor to threshold-aware GRINCH discussed in Section 4.4.2, however GRINCH-SHALLOW stores a forest of trees. These trees represent a partition of the leaves, where each subtree represents a cluster of its corresponding leaves. GRINCH-SHALLOW requires a single hyperparameter, τ, that represents a threshold used to define each tree in the forest it maintains. The linkage scores of *every* node in every subtree maintained by GRINCH-SHALLOW must be greater than or equal to the threshold.

After any modification of a node v, the linkage scores at all ancestors of v may change. Operating in a bottom-up fashion, if ever a linkage score that is below the threshold is encountered, the corresponding node and all of its ancestors are immediately deleted. This reduces the number of nodes stored and increases the number of subtrees.

In GRINCH-SHALLOW, the `xgraft` may still perform grafts across trees in the forest. A result of this is additional complexity in restructuring (i.e., `xrestruct`), when the two nodes that initiate the restructure are a part of separated subtrees. In this case, consider those two trees to be siblings under a hallucinated root and perform the `xrestruct` normally. After the procedure, delete the hallucinated root if its linkage score (and the linkage score of all of its descendants) do not exceed the threshold τ.

We note that GRINCH-SHALLOW can be applied to GRINCH as well as XGRINCH, as long as the desired result is a flat clustering (performed via thresholding) rather than a hierarchical clustering. In both cases, GRINCH-SHALLOW reduces the number of nodes in the tree. Especially when combined with XGRINCH to produce XGS, the resulting forest of trees can be short and wide which can significantly reduce the number of internal node updates required, thereby expediting runtime.

5.2 Errors Sources and Edit Placement

Recall that the collection of mentions from which a KB is built is typically produced via information extraction. For example, in a scientific KB, the author mentions may have been created by extracting biographical information from publications. Unfortunately, most information extraction techniques are prone to error. These errors affect downstream tasks like ER and canonicalization, culminating in KB errors. Unlike ER errors, KB errors that stem from noisy mentions are not alleviated by repartitioning the mentions (i.e, splitting or merging the inferred entities). Instead, the noisy attributes and relations must effectively be "overwritten" during canonicalization. Therefore, proper handling of a particular KB error is a function of the source of that error.

Previous work highlights the fact that KB users have significant potential to help correct KB errors [57]. However, as that work points out, all errors manifest to KB users similarly, i.e., they all appear to a user as missing and spurious attributes of inferred KB entities. Thus, it neither realistic to expect that users are able to identify error sources nor that they will encode errors sources in the feedback they submit in response. Since errors of various types must be handled differently, it is effectively up to the KB to "determine" the source of that error and act accordingly. Because of the ambiguity in such determinations, the KB should be able to easily make revisions as new evidence (i.e., data and feedback) elucidates the true sources of error.

In this section we discuss various error types and how they may be corrected. We highlight that the precise placement of an edit in the tree constructed by GRINCH determines its function. Moreover, we illustrate the degenerate behavior of GRINCH in a common case where many mentions are equally similar to each other. Furthermore, we show how XGRINCH naturally avoids this undesirable behavior by virtue of building trees with arbitrary branching factor.

5.2.1 Dual Functionality of Edits

We begin by presenting an example that highlights the close relationship between edit placement in a tree built by GRINCH and sources of KB errors. Consider Figure 5.4a, which depicts three mentions of the same ground-truth entity and their attributes. Mention x_3 contains attribute D, which is noisy. After canonicalization, which in this example is performed by set union, the inferred entity exhibits the attribute D, which is incorrect. Note that there are no ER errors in this example.

Assume that a user detects this error and contributes feedback, f_1, which has a packaging (Section 4.3.1.1) that contains the attribute A (e.g., an email address) and a payload containing the negation of the attribute D (e.g., an incorrect affiliation). Furthermore, assume that the model used by GRINCH assigns a score of negative infinity to any two subtrees where one contains an attribute and the other contains the negation of that attribute. Assume that f_1 is placed as a sibling of x_1, as a sibling of x_2, or as a sibling of their parent (perhaps after rotations). Then, partway through canonicalization, the root of the subtree containing x_1, x_2 and f_1 exhibits attributes $\{A, B, C, -D\}$, making it *incompatible* with x_3 according to the model, and leading to an incorrect split of the tree (Figure 5.4b). On the other hand, consider the case that f_1 is made the sibling of x_3. Then, after canonicalization, the attributes of the parent of f_1 and x_3 are $\{A, E\}$. In this case, no splits are incurred and the inferred entity has all of the correct attributes. A similarly positive outcome occurs if f_1 is made a sibling of the root of the tree.

This simple example highlights the tight coupling of feedback placement in a GRINCH tree and sources of error in KBs. In the example above, we assumed that the attribute D appeared in mention x_3 because of noise, and therefore, the *only* placements of f_1 that yield correct results are: as a sibling of x_3 or a sibling of the root. However, had we assumed that x_3 was a mention of a ***different*** ground-truth entity (and not that x_3 was noisy), then placing f_1 as a sibling of x_3 or the root yields an *incorrect* result; instead, it must be made a sibling of either x_1, x_2, or their parent. Interestingly, when f_1 resides deeper in the tree than

x_3, i.e., x_3 is a sibling of an ancestor of f_1, f_1 causes a split in the tree; otherwise, f_1 serves to effectively "overwrite" the incorrect attribute D. Generalizing this example reveals that an edit with a negatively-weighted attribute only "overwrites" the corresponding (positive) attribute if it is a sibling of a node that contains that attribute.

This example reveals the dual functionality of edits with packaging and payload, i.e., the same edit can be used to correct ER errors and noisy data errors. In addition to being powerful, this dual functionality also facilitates revision of error source determination. Concretely, if the KB initially believes that a particular piece of feedback aims to correct an error stemming from noisy data, the feedback can be placed accordingly in the tree. If new evidence later reveals that the feedback was contributed in response to an ER error, in theory, the KB may simply change the location of the feedback to produce a more favorable resultant tree; no changes to the raw, underlying feedback or mentions is necessary. As discussed above, the ability to easily revise determinations about error sources is the best that can be hoped for, since a unit of user feedback does not identify the source of the error it aims to correct.

5.2.2 Equivalent Subtree Reconstructions

One undesirable consequence of the GRINCH (but not XGRINCH) integration framework proposed so far emerges when considering subtrees that can be reconfigured into a variety of distinct and "equivalent" subtrees (under the particular linkage function being used). In these cases, a subtree's structure is *entirely* dependent on mention arrival order. Concretely, assume the model used by GRINCH computes similarity scores based on attribute overlap and assigns the same similarity to the pairs: $(\{x_1\}, \{x_2\})$, $(\{x_1\}, \{x_3\})$, $(\{x_2\}, \{x_3\})$, and $(\{x_1, x_2\}, \{x_3\})$. Simpler models (e.g., decision sets [60]) could reasonably exhibit this property and are arguably likely to be used in practice because they are highly interpretable and easier to debug that more complex models. For example, in a bibliographic KB, a reasonable model may assign maximal similarity to two mentions that share the same

email address (in the example above, assume that A represents an email address). Were this the case, then the tree structure in Figure 5.4a has multiple equivalent reconstructions. Specifically, a structure where x_1 and x_3 are siblings and x_2 their aunt is equally "good," according to the model, and is the result of applying GRINCH to a particular arrival order of the mentions.

A problematic side effect of such a situation is that the placement of feedback in such a subtree is also determined by the time at which it arrives. For example, in the Figure 5.4a, it is reasonable to assume that f_1 (with attribute $-D$) in its payload is equally similar to all leaves according to the model. Therefore, its placement in the tree is arbitrary (and unpredictable). This is suboptimal since the location of f_1 determines its function (either as an indicator of an ER error or noisy data); a decision that should not be made arbitrarily. Moreover, this outcome is at odds with the central theorem of GRINCH, which states that the algorithm is robust to data arrival order (Theorem 1).

For subtrees with equivalent reconstruction, collapsing internal structure with XGRINCH eliminates unpredictable feedback placement. This is because all nodes in those subtrees reside under the same parent. This, in turn, improves consistency and robustness with regards to the dual functionality of edits, because negations stored in a payload do not affect their siblings.

5.3 Remarks on XGS for Recovering from Noise

While XGS (and even XGRINCH) help to make the KB more resilient to the placement of feedback in specific cases (namely, when there are groups of mentions that are all equally similar), they do not completely solve the issue of determining the sources of various KB errors. This is due to the facts that:i) users do not communicate their beliefs about error sources in attribute feedback, ii) even if users could do so, it is not clear that their beliefs would be correct, and iii) users have no knowledge of the hierarchy over mentions, and thus cannot design feedback that perfectly solves the errors they identify. However, these

issues are inherent when all errors manifest similarly (i.e., to a user, they all appear to be either missing or spurious inferred entity attributes) and identity uncertainty of mentions and feedback must be resolved. Moreover, it seems likely that any comprehensive solution must either severely limit the set of valid feedback, or must directly involve the user. Indeed, this observation leads to the potential for alleviating the issues discussed via a better user editing interface (UI). Such a UI could include multi-step user interaction, where the UI's "actions" are based on computations of how specific feedback might effect the KB. Sensing that feedback provided by the user would cause a repartition of the mentions, the UI may prompt the user to verify subsequent mergers and splits. If the user responds affirmatively, the UI could determine the attributes included in the feedback, and whether they should be stored in its packaging of payload. Exploration of better UIs for soliciting user edits is a fertile area of research, but is beyond the scope of the current work (Chapter 6).

Despite these issues, we highlight the benefits of our proposed approach, especially in comparison to alternatives. First, our approach allows for some semblance of maintaining uncertainty about error sources: even if the initial placement of feedback does not match the source of the error it is meant to correct, as more evidence arrives (i.e., additional feedback and mentions), placement of the feedback can be updated. This is in contrast to a more rigid approach that predicts an error source when feedback is contributed; a decision that may be difficult to change when more evidence arrives. Second, by allowing payload attributes to effectively be ignored until after initial canonicalization, our approach implicitly abides by the principal that a negation present in a group of equally similar mentions (e.g., in a tree constructed by XGRINCH) usually is indicative of noise rather than an ER error. That is, if one mention in a group of siblings contains the negation of an attribute that is present in another, it is more likely that the negation is an attempt to correct noise, rather than an ER error. However, after canonicalization, the negation can appear at ancestor representations, which deters mergers with other nodes that violate the negation.

5.4 Experiments: `OpenReview.net`

In this section, we turn our attention to evaluations of our proposed KB integration framework in a real-world setting: `OpenReview.net` (OpenReview). In particular, we use XGS to perform ER across all records stored throughout the entire website, including human edits. We measure the size of the trees produced by GRINCH, GRINCH-SHALLOW and XGS. A subset of the ER predictions are manually labeled by the OpenReview team, which serves as ground-truth. We use this ground-truth to evaluate the utility of our framework, including edit mention structure, which is comprised of a packaging and payload. We begin by describing the `OpenReview.net` landscape and how XGS is applied, and then present quantitative results related to XGS's utility to the OpenReview platform.

5.4.1 Background: `OpenReview.net`

OpenReview is a conference management platform. Conference program chairs can use OpenReview to solicit submissions to their conferences, invite reviewers, match reviewers to papers, facilitate paper reviewing, and more[2]. To support its functionality, OpenReview maintains a KB of scientists. For each scientist, the KB stores the scientist's names, emails, home pages, related web links (e.g., DBLP profile or wikipedia page), academic relationships, expertise (stored as keywords), and publications. The KB is leveraged by conferences to select area chairs and reviewers, detect conflicts of interests between reviewers and submission (using affiliation history and academic relationships), and model reviewers' affinities for paper submission (using their known expertise and previous publications). OpenReview users are able to browse the KB via *profile pages*, one per scientist.

The scientist KB is ever-growing. In particular, new information can be added to the KB in two ways: i) users may add/remove data from their own profiles (i.e., human edits), and ii) the OpenReview team may upload new data and assign it to an existing (or new) profile.

[2]See `https://openreview.net/about` for more information.

Since, users may only edit their own profiles[3], the data they contribute is seldom incorrect[4]. In contrast to users who add a handful of edits at a time, uploads from the OpenReview team often add large amounts of data to the KB. For example, to date, the OpenReview team has added more than 100,000 publications stored in DBLP[5] to the KB of scientists (they have also added data from other sources). During a large upload, the OpenReview team uses their own collection of heuristics to perform ER. For each piece of data, these heuristics attempt to correctly attribute that data to a scientist already stored in the system's KB, or determine that a new profile should be created to house the data. Performing ER in this setting is challenging and results in errors. Crucially, all data—i.e., user provided edits as well as data uploaded by the OpenReview team—are stored as immutable[6] records that can be thought of as mentions.

5.4.2 Dataset: The European Conference on Computer Vision

OpenReview was chosen as a platform to help manage the 2020 edition of the European Conference on Computer Vision (ECCV). Prior to the conference, the OpenReview team performed a bulk upload of a subset of DBLP publications, where the subset included authors and reviewers from previous editions of ECCV and related conferences. The goal of the upload was to pre-populate the profiles of authors and reviewers before ECCV so that OpenReview would be better equipped to: i) model affinity between reviewers and submissions, and ii) facilitate the discovery of conflicts of interest between reviewers and

---

[3] As of May, 2020 users may only edit their own profile. However, there are plans to allow users to edit a wider collection of profiles, e.g., *any* profile. This presents a more difficult setting in which we expect their to be a larger number of data entry errors. Importantly, our proposed framework is designed to be effective in this setting.

[4] There are a handful of cases of incorrect data entry from users. We hypothesize that this is a result of users either "testing" the system or trying to resolve coreference mistakes.

[5] https://dblp.uni-trier.de/

[6] The OpenReview team *may* modify a specific record, but this is rare. By "immutable", we mean that once a record is added to the system, it is never overwritten or deleted by a user edit or other data upload. This is an instance of treating edits as mentions, that we advocate for earlier in this thesis.

	OpenReview	ECCV Subset
Number of mentions	663,957	59,848
Number of edits	25,411	14,787
Number of profiles	202,463	8,193

Table 5.1: Statistics of the full OpenReview.net KB as well as the subset corresponding to ECCV.

submissions. Both are crucial in matching submissions to reviewers. Uploaded records were either added to existing profiles or caused the creation of new profiles via the team's ER heuristics.

Afterward, XGS was run on the entirety of the OpenReview KB, including all previous human edits. The result was mention-level ER predictions. In detail, a prediction assigns each record—i.e., all human edits and other data in the system—to an *inferred entity*. During ER, inferred entities are represented as arbitrary ids, and any two records assigned to the same inferred entity are considered coreferent. The predictions can be interpreted as suggesting either: a merger among a collection of profiles, a split of one profile into multiple sub-profiles, or a combination of both. Desiring high precision entity resolution, the OpenReview team proceeded to manually inspect the XGS predictions and label those that were correct *at the entity level*, i.e., they used XGS's predictions to inform which current *profiles*—not mentions—should be merged together or split. Table 5.1 contains statistics describing the OpenReview KB as well as the OpenReview team's manual curation effort.

We use this hand curated list of mergers to evaluate our proposed framework for integrating human edits with a KB under identity uncertainty. However, we raise a handful of important considerations that must be kept in mind when analyzing our experimental results.

1. Due to resource constraints, the team focused only on candidate mergers. Therefore, we expect the number of false negatives (i.e., necessary mergers that were not predicted by XGS) to be low.

2. Similarly, evaluations using the data do not substantially address the quality of XGS's predictions corresponding to splitting profiles.

3. The manual labels may contain errors. However, we expect the number of incorrect mergers and missed mergers (among the ECCV subset) to be small, if non-zero.

5.4.3 Experimental Setup

We use the labeled dataset described above to evaluate XGS, our proposed strategy of representing edits as mentions, and structuring edits with packaging and payload. XGS requires a entity-level similarity function. In our experiments, we test two models: the **lenient** model and the **strict** model. The difference between the two is how they handle human edit *negations* (Section 4.3.1), i.e., edits that express that a certain attribute is *not* true (for example, an edit expressing that a scientist was never affiliated with a particular university). The lenient model down-weights the scores between two groups of mentions where one group expresses some attribute, and the other expresses the negation of that attribute. The strict model predicts that two such groups of mentions are not coreferent, without regard for any other evidence. Both models are constructed by hand and resemble linear models[7] so that their decisions are interpretable and easy to debug, which are both crucial properties for a model deployed within OpenReview.

5.4.3.1 Evaluation

We evaluate XGS and our edit integration strategy using F-Score, where the ground-truth is comprised of the manual mergers labeled in the ECCV subset of OpenReview (Table 5.1). We report both *mention-level* and *entity-level* F-Scores. Mention-level statistics are computed using the standard protocol for computing pairwise F1. Entity-level statistics are computed by measuring F1 at the entity level. Specifically, at the entity level, a true positive

[7]Both models are linear except that they may return early if a telling conditions holds (e.g., two edits that were contributed by the same user to that user's profile page are considered coreferent).

describes a case in which XGS correctly predicts *all* mentions of an inferred entity, $\hat{e}_1$, should be merged with *all* mentions of inferred entity $\hat{e}_2$. In this way, entity-level statistics are much less forgiving than mention-level statistics. However, they reflect the OpenReview Team's data curation efforts, which entailed manual merging at the profile (i.e., entity) level. Additionally, entity-level statistics better reflect user experience than mention-level statistics. A single mention split from other coreferent mentions incurs a relatively large entity-level F-Score penalty, but a relatively low mention-level F-Score penalty. In OpenReview, such an error manifests to the user as *two separate scientists* (i.e., profiles); arguably a large error that, at the time of writing, requires intervention from the OpenReview Team to fix.

5.4.3.2 Constructing Packaging and Payload

When a user submits an edit via the profile UI, a corresponding edit must be created which contains a packaging and a payload (Section 4.3.1.1). In our experiments, by default, all existing information on a profile page is included in the packaging of an edit. Additionally, non-negation edits (e.g., a user adding an email address to their profile) are stored in the packaging. Negations (e.g., a user *removing* an email address from their profile) are stored in the payload with a weight of -1. This biases XGS to treat edits containing negations as correcting noisy data before consideration as corrections to coreference resolution (discussed further below, in Section 5.2.1).

5.4.4 Experiment 1: Size of Internal Structure

We begin our empirical analysis by measuring the size of the trees constructed by GRINCH, GRINCH-SHALLOW and XGS when run on the entirety of OpenReview.net, including user edits. We measure the number of nodes because the OpenReview team desires that our integration framework create the fewest number of additional records in their database as possible. Prior to running XGS, we assign each mention to at least one (but potentially many) canopies [100]. We run XGS on *blocks* of 100,000 mentions. For a canopy, c, all mentions that are members of c appear in the same block. The results appear

	GRINCH	GRINCH-SHALLOW	XGS
Nodes	1,327,913 (-0.0%)	1,123,938 (-15.36%)	914,087 (-31.16%)
Internal nodes	663,956 (-0.0%)	459,981 (-30.72%)	250,130 (-62.33%)

Table 5.2: **Tree Size.** Size of trees constructed by three ER methods on the OpenReview KB. Numbers in parentheses represent relative reduction in size compared to the tree built by GRINCH (i.e., full binary tree).

in Table 5.2. We observe that XGS reduces the overall tree size (in terms of number of nodes) by more than 30% when compared to GRINCH. Note that this includes the mentions and the leaves of the tree, which none of the three methods ever consolidate. When only considering internal nodes, we calculate that XGS reduces tree size by more than 60% on the OpenReview KB.

5.4.5 Experiment 2: Edits as Mentions

Next, we investigate the effect of treating edits as mentions and allowing edits to participate in ER. To do this, XGS is run on the entire OpenReview dataset (including user edits) and evaluated against the labeled ECCV data. We compare XGS with both the **lenient** and the **strict** model to the KB before ER is run (**no coref**). This represents the state of the KB after data has been uploaded and incorporated via the OpenReview Team's heuristics. All edits are assigned to the profile on which they are contributed. Note that we do not apply XGS for the **no coref** method.

We also compare XGS run on the *non-edit* data in the KB (**w/o edits**). This resembles an approach which runs coreference to build profile pages, and then applies edits to those pages, but does not use the edits to further improve ER. Note that the lenient and strict models only differ in their handling of edits, and so they both produce the same results in this setting. For the sake of a fair comparison, edits are omitted when computing precision, recall and F1 for all approaches.

The results appear in Table 5.3. The results show that running coreference and treating the edits as mention achieves higher F-Score than applying the edits to profiles after coreference.

Eval.	Strategy ↑	Precision ↑	Recall ↑	F-Score ↑
mention level	no coref	**1.0**	0.839	0.912
	w/o edits	0.999	0.929	0.963
	lenient	0.998	**0.932**	**0.964**
	strict	0.999	**0.932**	**0.964**
entity level	no coref	0.0	0.0	0.0
	w/o edits	0.992	0.324	0.489
	lenient	**0.994**	**0.399**	**0.570**
	strict	**0.994**	0.398	0.569

Table 5.3: **Edits as Mentions.** A comparison if the initial state of the KB, coreference run *without* including human edits, and the edits-as-mentions approach advocated in this thesis.

The improvement is particularly dramatic at the entity level (7%). Both the lenient and strict models perform similarly. Note that at the entity-level, the **no coref** strategy achieves 0.0 F-Score. This is because, at the entity-level, each profile is a singleton, and therefore with no merging at the entity-level, **no coref** predicts 0 true positives.

5.4.6 Experiment 3: Packaging and Payload

Next, we investigate our proposed packaging and payload edit structure. Recall that attributes included in user edits may be stored in either the packaging or the payload (Section 4.3.1.1). Attributes in the packaging are used to compute model scores between groups of mentions. During canonicalization, any attributes stored in a mention's payload are added to its parent's packaging. Together, packaging and payload are sufficient for handling missing data, data errors, and coreference errors (Section 5.2.1). However, users do not specify which attributes of the edits they provide should be included in the packaging and which attributes should be included in the payload.

In this experiment, we evaluate 3 methods of assigning attributes to packaging and payload. First, we test the **pack only** strategy, which stores no attributes in the payload. This corresponds to a naive implementation of edits as mentions. Next, we test two more distinct approaches; both store all attributes of the profile being edited in the packaging. The **negative pay** strategy stores all negations contained in a user edit in the payload (Section

Eval.	Model	Edit	TP ↑	FP ↓	FN ↓	F1 ↑
	L	pack	959638 (+0.00%)	**1308 (+0.00%)**	73277 (+0.00%)	0.963
	L	neg.	**962880 (+0.34%)**	1326 (-1.38%)	**70035 (+4.42%)**	**0.964**
ment.	L	mod.	961794 (+0.22%)	**1308 (+0.00%)**	71121 (+2.94%)	**0.964**
level	S	pack	961512 (+0.20%)	**1308 (+0.00%)**	71403 (+2.56%)	**0.964**
	S	neg.	962652 (+0.19%)	1326 (-1.38%)	70263 (+1.60%)	**0.964**
	S	mod.	961566 (+0.01%)	**1308 (+0.00%)**	71349 (+0.08%)	**0.964**
	L	pack	1161 (+0.00%)	**6 (+0.00%)**	1758 (+0.00%)	0.568
	L	neg.	**1166 (+0.43%)**	7 (-16.7%)	**1753 (+0.11%)**	**0.570**
entity	L	mod.	1163 (+0.17%)	**6 (+0.00%)**	1756 (-0.17%)	0.569
level	S	pack	1158 (-0.26%)	**6 (+0.00%)**	**1761 (+0.11%)**	0.567
	S	neg.	1163 (-0.26%)	7 (-16.6%)	1756 (-0.17%)	0.569
	S	mod.	1160 (-0.26%)	**6 (+0.00%)**	1759 (-0.06%)	0.568

Table 5.4: **Packaging and Payload.** Effects of various constructions of packaging and payload on true positives (TP), false positives (FP) and false negatives (FN) and the corresponding percent improvement over the **pack only** strategy with the lenient model.

5.4.3.2). In the **modify pay** strategy, if, for a particular field, there is both a negation and a new value (e.g., a user removes a existing email from their profile and contributes a new one), then both are stored in the payload; otherwise, the new attribute and/or negation are stored in the packaging. In this way, the **modify pay** strategy applies a heuristic that tries to identify instances of users *correcting data errors*, e.g., misspelled email addresses, incorrect affiliation years, etc. These errors should be remedied through "overwriting" rather than modifications of coreference, and therefore, when users add or *modify* (rather than remove) attributes from their profiles, these attributes are stored in the payload where they have less chance of affecting coreference decisions.

Table 5.4 compares the performance of the 3 strategies. While the 3 strategies perform similarly in terms of F-Score on the ECCV dataset, the **modify pay** strategy achieves the largest number of true positives and the smallest number of false negatives. Moreover, the **pack only** strategy achieves the lowest number of true positives and the highest number of false negatives. The results suggest that there is utility in structuring edits with packaging and payload, rather than including all edit attributes at packaging.

5.4.7 Experiment 4: Must-link and Cannot-link Constraints

In this dissertation we proposed attribute editing, in which users provide edits to canonicalized, inferred entity profiles (Section 4.2.3). These edits contain new attributes as well as negations of existing attributes. Another type of user edit, which are more common in the ER and clustering literature and which we previously discussed, are must-link and cannot link constraints. Attribute editing is a strict generalization of must-link and cannot constraints. That is, must-link and cannot-link constraints can be simulated via attribute editing. However, the reverse is not true: classic must-link and cannot-link constraints cannot be used to encode all attribute edits.

As an empirical proof we again turn to the ECCV dataset. Recall that the OpenReview team used XGS predictions to inform which profiles should be merged. In manually deciding which profiles should be merged, the OpenReview team effectively created edits that could be interpreted as must-link constraints among profiles. Morevoer, after the OpenReview team performed the mergers, they were alerted (usually by the profile owners) that some merges were incorrect. The team proceeded to split the profiles back into their constituents that existed before merging. These splits can be interpreted as cannot-link constraints among profiles.

We simulate must-link and cannot-link constraints via attribute editing as follows. For each merger, we create a mention that stores *merge ids* of all mentions (including edits) associated with the profiles being merged. Moreover, we modify the model used by XGS (i.e., lenient and strict) to return a high score for any two groups of mentions such that one group contains mentions whose ids are stored in the merge ids of other. For a split of one profile into multiple constituent profiles, we create one mention per constituent. Each mention stores the merge ids of the mentions associated with its constituent, as well as the *split ids* of the mentions in the profiles it must be split from. We then modify the model so that it gives a low score if to two groups of mentions such that one group contains mentions whose ids are in among the split ids of the other. If two groups contain both corresponding

split and merge ids, the groups are deemed to be not coreferent. We run a final proof of concept with the modified model and and find that we achieve perfect F-Score, as expected. Most important, this shows that flexibility of attribute editing, and also shows how mergers and splits performed by the OpenReview Team can be encoded as mentions, and included among the rest of the evidence during coreference resolution.

5.4.8 Discussion

The experiments in this section represent an application of our proposed framework—which includes representing edits as mentions with packaging and payload—in the context of a real-world system: `OpenReview.net`. Our results tell a consistent story: leveraging user edits during ER improves the mention-level and entity-level F-Score of the KB on a small labeled data subset. Moreover, we observe small, but consistent, improvement in F-Score when edit contents are partitioned into packaging and payload. In practical terms, using XGS helped the OpenReview team merge 8,193 incorrectly split profiles into 4,303 profiles.

In our second experiment (Section 5.4.5) we notice is that effect sizes are larger at the entity-level than the mention level. Specifically, when edits are represented mentions and used during ER, we observe a 7% improvement in F-Score at the entity-level. This is much larger than the same statistic at the mention level (0.1% improvement). The reason for this discrepancy is that many profiles are split such that many components of the split only include a handful of mentions. This leads to relatively small penalties at the mention-level but large penalties at the entity-level. However, in practice, this type of error results in a user having multiple corresponding profiles, which, we argue, is a significant issue.

This trend is also apparent in the packaging and payload experiment (in terms of F-Score), yet is more understated. Here, we see that the naïve strategy of including all user edit contents in the packaging yields lower F-Score numbers than splitting the attributes into

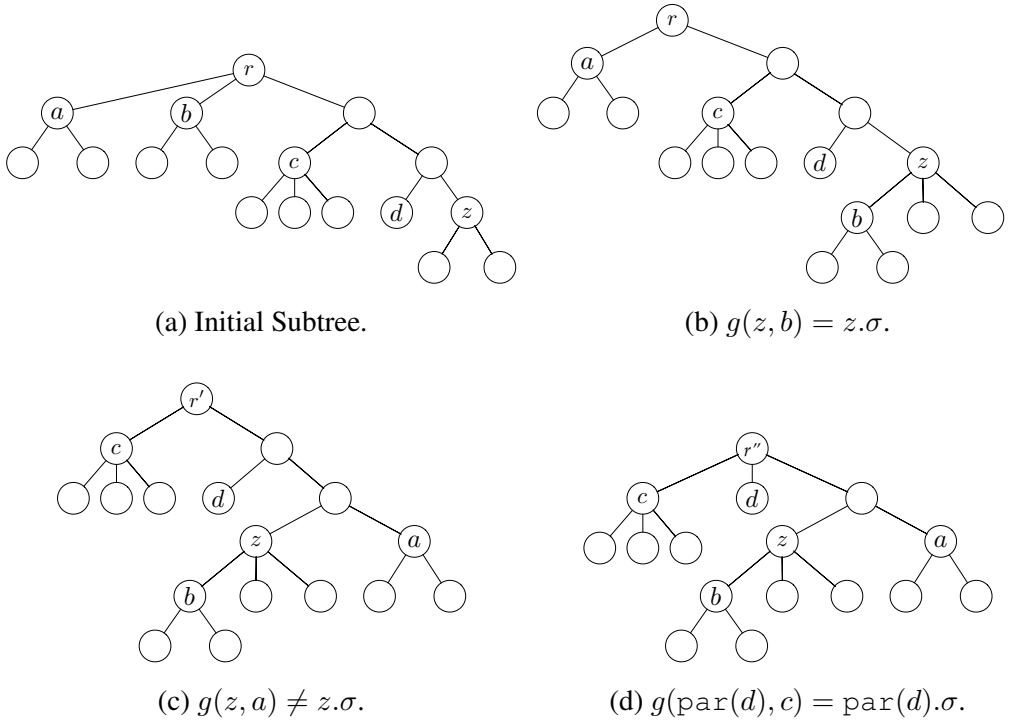
packaging and payload. Even though the effect size is small, this result suggests that there is a benefit to (deliberately) partitioning edit contents.

We believe that the effect sizes are small for a handful of reasons. First, note that we evaluate our approach using F-Score, which measure effectiveness of ER. In particular, our measurements correspond to how much better user edits make XGS at ER. What this *does not* include are: edits that contribute missing attributes on a profile or correct incorrect attributes on a profile, but do not affect coreference. For example, an edit specifying which of a user's email addresses is their preferred email address typically does not affect ER. Theses types of edits make up a significant portion of the edits in the ECCV subset, meaning that effect sizes—measured in terms of ER—are expected to be small.

Additionally, we note that in the ECCV dataset, edits are generated by logged in users about *their own* profiles. With this knowledge, the models used by XGS (i.e., lenient/strict) always consider edits generated by the same user as coreferent. Thus, the edits do not exhibit identity uncertainty, which makes ER considerably easier. For this reason, and for the sake of fair comparison with methods that do not leverage user edit, the edits are excluded when measuring F-Score (but included in ER), decreasing the total number of mentions in the evaluation. Since user edits can be assumed to always be assigned to the correct profile, the primary benefit of using the edits (with respect to our F-Score evaluation) is to resolve identity uncertainty associated with data uploaded by the OpenReview team. This includes instances mentions there were originally not-included in a users profiles, and instances of mentions being incorrectly included on a user's profile. As the results show, performing coreference at the mention-level without using edits achieves relatively high F-Score (96%), which leaves little room for improvement once edits are integrated. Despite this, we still see improvements when leveraging the user edits.

Finally, we remark that the ECCV dataset is relatively small and only contains profile mergers (as opposed to mergers *and splits*). Unlike many author coreference datasets [21, 42], the ECCV data was not designed to be ambiguous, i.e., a difficult dataset in which to

perform ER. Instead, it reflects a real instance in which ER is necessary, which includes many non-ambiguous author names. Despite the above considerations, the experimental results are evidence of the utility of our proposed approach.



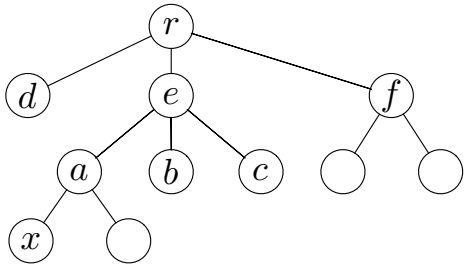
(a) Initial Subtree.

(b) $g(z, b) = z.\sigma$.

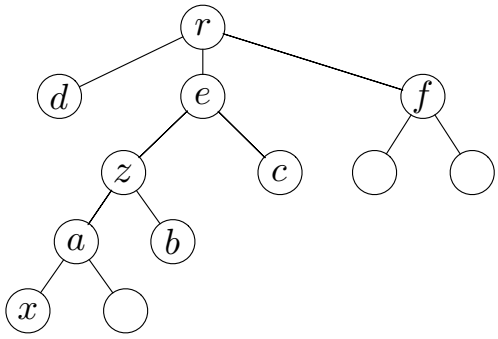
(c) $g(z, a) \neq z.\sigma$.

(d) $g(\texttt{par}(d), c) = \texttt{par}(d).\sigma$.

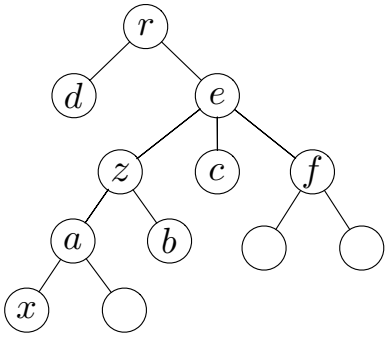
Figure 5.2: **`xrestruct`.** Figure 5.2a shows a subtree immediately after a sibling of z has been moved elsewhere (e.g., as a child of b). First, collect the siblings of z and the siblings of its ancestors, i.e., a, b, c, and d. In this example, b is the highest scoring with z (according the model g). Since $g(z, b) = z.\sigma$, b is made a child of z (Figure 5.2b). Afterward, the process is repeated from the parent of the best sibling, i.e., $\texttt{par}(b) = z$. The new highest scoring node with z is a, but $g(z, a) \neq z.\sigma$ so a is made a sibling of z (Figure 5.2c). The process is repeated at the parent of the best sibling, i.e., $\texttt{par}(a)$. The new highest scoring node is d (among c and d). Since $g(\texttt{par}(a), d) \neq \texttt{par}(a).\sigma$, but $\texttt{par}(a)$ and d are already only siblings, the tree remains unchanged (not depicted). Finally, the process is repeated from $\texttt{par}(d)$. Since $g(\texttt{par}(d), c) = \texttt{par}(d).\sigma$ (Figure 5.2d), c is made a child of $\texttt{par}(d)$. Afterwards, `xrestruct` terminates.



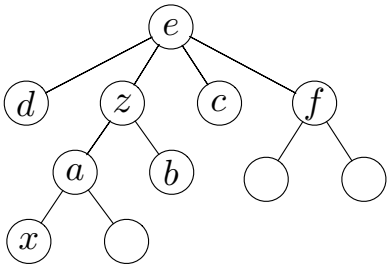
(a) Input x. `xupdate` from a.



(b) $a.\sigma > g(a,b) > g(a,c)$.

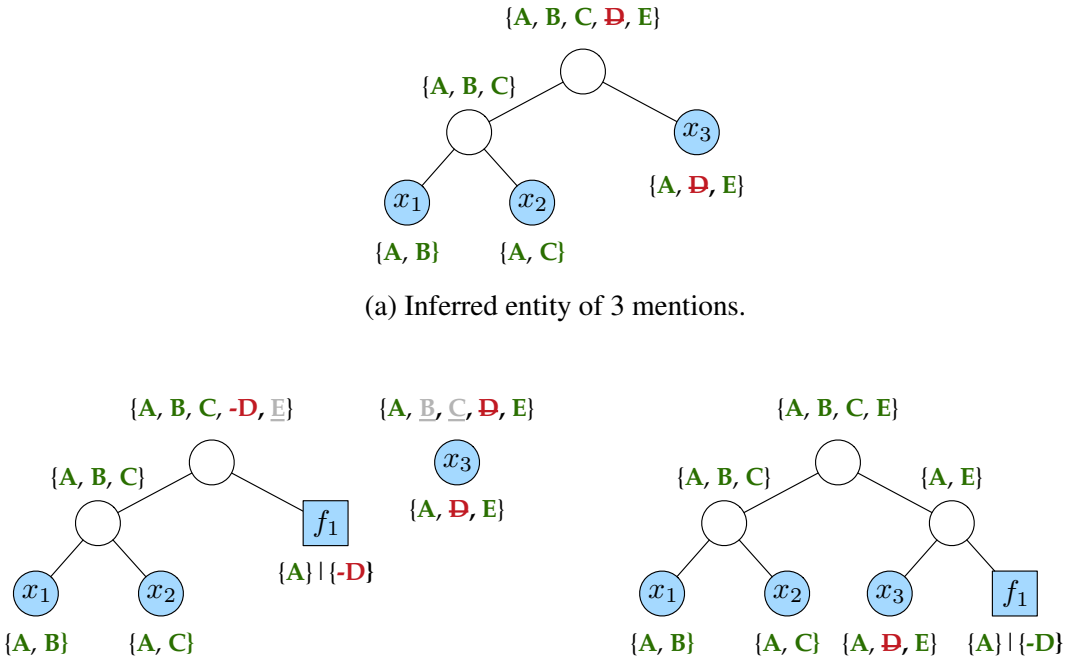


(c)
$z.\sigma > g(z,c); \quad e.\sigma = g(e,f) > g(e,d)$.



(d) $e.\sigma = g(e,d)$.

Figure 5.3: **`xupdate`.** Let x be the newly inserted node, then an `xupdate` begins from its parent, a. Find the highest scoring sibling with a under model g—in this example, b. If $a.\sigma > g(a,b)$, then make a and b siblings (Figure 5.3b). Continue from $z = \texttt{par}(a)$, the (new) parent of a. Again, $z.\sigma > g(z,c)$ so z and c remain (binary) siblings. However, when repeating the procedure from e, f is made a child of e since $e.\sigma = g(e,f)$ (Figure 5.3c). Finally, d is also made a child of e since $e.\sigma = g(e,d)$ (Figure 5.3d). This shows how `xupdate` maintains Invariant 1.



(a) Inferred entity of 3 mentions.

(b) Suboptimal edit placement.

(c) Optimal edit placement.

Figure 5.4: **Effect of edit placement.** Circles and squares at the leaves of the tree represent mentions and edits, respectively. All mentions and feedback refer to the same ground-truth entity (represented by blue fill). Letter-filled, brackets below/above nodes represent raw and canonicalized attributes; the edit, f_1, stores A in its packaging and $-D$ in its payload. Green, gray and red letters represent correct, missing and incorrect attributes, respectively. Figure 5.4a illustrates a subtree before an edit is provided. The corresponding inferred entity has a single incorrect attribute coming from noise in x_3. Figure 5.4b shows one placement of the edit (f_1), which results in two inferred entities (incorrect). Figure 5.4c shows a second placement of the edit, which alleviates initial incorrect attribute.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

This thesis studies methods for leveraging user feedback during KB construction, especially for KBs whose mentions exhibit identity uncertainty. To begin, we outlined a number of desiderata for a user feedback integration framework (Chapter 2). These desiderata led us to develop GRINCH, an incremental ER algorithm that performs both local and *global* rearrangements to support splitting and merging of mention clusters in response to new data (Chapter 3). We prove that GRINCH is theoretically well-motivated and that it works efficiently and effectively on both author coreference and general clustering benchmarks.

Next, we address one of the key challenges of user feedback integration: feedback that exhibits identity uncertainty (Chapter 4). In light of this challenge, we propose to represent user feedback *as mentions* and allow the ER algorithm to determine to which KB entity each piece of feedback refers. We also identify the challenge of determining the source of each KB error. To this end, we propose a representation of user feedback that is comprised of *packaging* and *payload*, which allows feedback to facilitate recovery from multiple error types. In author coereference experiments with synthetic user feedback we show that treating feedback as mentions with packaging and payload tends to lead to the fastest recovery of the ground-truth partition (when compared to other treatments and representations of user feedback).

Finally, we study user feedback integration in the `OpenReview.net` system (Section 5.4). Due to the scale of the OpenReview database, as well as the requirement to run in a small amount of memory, we develop XGS (Chapter 5), a variant of GRINCH that builds a forest of trees having arbitrary branching factors. We use XGS to run ER on the entire

OpenReview dataset and then use a labeled subset of the data (i.e., the ECCV subset) to evaluate our approach. The results reveal that in comparison to GRINCH, XGS builds a tree on the OpenReview data that exhibits a 62% reduction in the number of internal nodes required. Consistent with previous experiments, we find that representing feedback as mentions with packaging and payload leads to gains in the F-Score of entity resolution.

6.1 Directions for Future Work

This thesis helps to initiate a path forward for leveraging user feedback to help make KBs more complete and accurate. Despite our positive experimental results, this topic merits futher investigation. In the following subsections we discuss 7 distinct avenues of related future research.

6.1.1 Alternative Feedback Styles

In this thesis we develop and focus on *attribute feedback* (Section 4.2.3), which grants users the ability to provide feedback at the entity-level. In Section 5.4, we also briefly describe how attribute feedback is a generalization of traditional pairwise-mention feedback. However, other styles of feedback exist, and new modes of user interaction with a KB come to mind easily. Below, we describe two alternatives to attribute feedback, but we also stress that the space of user feedback formats is unbounded and that each KB may benefit from a distinct style of interaction.

As a first example, we raise Snorkel, in which users write heuristics that are automatically combined with one another to help label data [90]. Unlike Snorkel, these heuristics could be used instead at inference time (i.e., during ER and canonicalization). In OpenReview.net, users could provide similar heuristics, expressing their beliefs about things like: the number of simultaneous affiliations a person is likely to have, the length of a typical career in research, the number of distinct areas of expertise one person is likely to have, etc.—similar in flavor to generalized expectation [74]. These heuristics could even be easily supplied

by the KB maintainers (e.g., the OpenReview team), who spend a considerable amount of time with the underlying KB data, but may have little expertise in KB construction techniques. In KBs that exhibit identity uncertainty, it is easy to see that these heuristics could inform both ER and error correction. However, many related items require further investigation: whether it would be most effective to encode these heuristics as mentions, how they could be consumed by XGS, and the extent of the change in our proposed feedback integration procedure that they would require. In a sense, these heuristics would exhibit identity uncertainty: while a good heuristic might hold for many KB entities, there would inevitably be a few for which it does not hold. Additionally, these heuristics may lead to new challenges in determining the source of KB errors (i.e., either ER or missing/corrupted data).

Another style of user feedback includes prompts from the system for users to input data. For example, we might consider the KB soliciting currently logged in users who are browsing the KB to verify certain facts as True or False; or soliciting users to contribute specific pieces of data about others. We even envision scenarios where edits are supplied *implicitly* rather than explicitly. For example, consider the process of recommending reviewers during peer review. During this process, area chairs identify reviewers because of their alleged area of expertise. Each reviewer identified in this way could be treated as a raw piece of feedback that names a reviewer and their expertise (and potentially other attributes). Feedback collected in this style—or more explicitly—could be easily represented as mentions and used with XGS. However, feedback submitted by a user about others is likely to induce more challenges related to identity uncertainty inherent in that feedback.

6.1.2 Models for ER

In our experiments with the OpenReview.net KB (Section 5.4) we use two, simple, hand constructed models (i.e., not automatically trained). The reason for this is that the relationship between XGS and the model being used is complex and difficult to reason

about. Utilizing a simple model makes diagnosing errors easier; a crucial step before model development in a production environment. Similarly, it is desirable to have some guarantee that the model will behave in expected ways. For example, if an OpenReview user identifies their DBLP page url, XGS should consider merging all corresponding DBLP entries with that user's profile; the opposite certainly should not occur. This is easy to guarantee with some handmade models and more difficult to enforce for learned models. Given the widespread success of trained models for ER, and especially author disambiguation, we are confident that a trained model could outperform our hand constructed models [42, 104, 21, 103, 120, 71, 110, 112, 127, 6]. However, it is unclear whether or not we could endow such a trained model with our inductive biases, and, even if we could do so successfully, if such a trained model would be easy to diagnose and modify such that it does not make embarrassing errors.

Related work exists for the above concerns. Classic methods like generalized expectation and posterior regularization attempt to encourage models to take up user specific inductive biases [74, 34]. There has been significant recent interest in training *interpretable* models via machine learning techniques [106, 60, 47]. Therefore we believe that developing methods for training models that obey certain mandated behaviors and that are also easy to diagnose represents a fruitful path for future work.

Finally, in long-lived KBs like OpenReview, there is a pressing need to update models as vast amounts of new data—and especially labeled data—are ingested into the system. Thus far, all of our experiments and setup assume that the model being used by XGS is static, which is likely unrealistic in the long-term. However, changing the underlying ER model could result in significant changes to the trees built by XGS, thereby negatively effecting user experience. Therefore, we also envision a need for methods of updating the model used by XGS in a way such that the downstream changes to the OpenReview KB are predictable, and that the extent of these changes is well-understood, and perhaps, bounded.

6.1.3 Sources of Error in KBs

One of the primary challenges that arises in integrating user feedback with KBs under identity uncertainty is correctly reasoning about the sources of various KB errors. For example, in OpenReview, when a user provides an edit claiming that they were never affiliated with a particular institution, in order to make the appropriate correction, the system must determine whether the mistake was caused by ER or a noisy mention. While we develop the packaging and payload representation of edits to be flexible enough to remedy both types of errors, this thesis does not definitively solve the issue of error source determination.

One instance of previous work studies the ability to diagnose systemic errors in KBs [115]. This approach proves useful for KBs with identity uncertainty as well. However, unlike this approach, we envision that certain types of errors and certain types of user feedback may be indicative of the types of corrections necessary. While we can envision certain multi-step interactions between the KB and a user to better understand the nature of certain errors (e.g., *We see that you've indicated your email address is incorrect; is this likely a typo, or does this email belong to someone else?*), this style of solution is cumbersome in that it requires significant engineering and testing, as well as time and attention from a logged in user. Further investigations of error soure determination in KBs that exhibit identity uncertainty is likely a fruitful area of future work.

6.1.4 Feedback Integration in Open Domain KBs

Our study has largely focused on KBs that posses relatively structured schemas for entities. For example, in OpenReview, scientist entities have affiliations, biographical information, specific relationships they can participate in, etc. In contrast, many KBs are *open domain*, where, for example, surface forms occurring between two entities in text may be considered a relation [9, 124, 30].

Open domain KBs that exhibit identity uncertainty pose additional challenges for user feedback integration. As one example, in these KBs, canonicalization plays a more significant role, since it requires that the KB determine which distinct text strings express the same relationship (or attribute). First, this creates the additional challenge of detecting when user feedback attempts to correct ER, noise, *or canonicalization*. Second, corrections to canonicalization should be applied *globally*. For example, user feedback expressing that the relationships `was born in` and `birthplace of` are semantically equivalent should be applied throughout the KB, and not only to the KB entity who is the target of the feedback. Representation of such user feedback and its application represent additional areas of future study.

Just like in the KBs we study, in open-domain KBs, users may provide feedback indicating *missing* relationships and attributes. More so than in our work, missing data errors may be the result of errors in *information extraction*, an error source we do not consider. Determining how to most effectively leverage user feedback to remedy errors in information extraction also represents an avenue for future work.

6.1.5 Attributes with Continuous Representations

In our experiments, KB entities participate in relationships and possess attributes that are represented as discrete objects. However, many KBs represent relations and attributes in a continuous manner [93]. This scenario opens many questions like: how are attributes with continuous representations shown to the user, what mechanisms should be used to solicit user feedback in these cases, and how should the feedback be applied. For example, it is conceivable that a scientist's expertise would be represented as a dense vector, and then displayed to the user as a ranked list of key words, or keywords with associated strengths. A user may desire to add/remove some of these keywords, increase/decrease their strengths, rank the keywords in an alternative order of descending strength, etc. Additional considerations immediately come to mind with respect to *negations*: how should feedback

claiming that a scientist is *not* an expert in a particular field be represented and how should it be applied? Challenges also lie in determining how such feedback is canonicalized. We note that feedback to continuous KB objects may still exhibit identity uncertainty and may be difficult for ER to integrate.

6.1.6 Real Time Feedback Integration

GRINCH and XGS are incremental algorithms by design. This is because user feedback naturally arrives continuously and in an online fashion. However, our experiments are neither performed in a real-time setting, nor do they explore all of the challenges of real-time user interaction.

We hypothesize that real-time integration may encourage users to supply additional feedback. This is because with real-time integration, users can see the effects of their feedback immediately. For example, in OpenReview, after a user supplies a missing affiliation, a real-time run of XGS could discover ER errors, resulting in additional publications appearing on that user's profile. If any of these publications were not written by the user, the user might supply additional feedback in response.

While XGS is designed to be efficient enough for real-time integration, other engineering concerns exist, e.g., locking. Since XGS is able to rearrange the hierarcichal clustering of the mentions both locally and globally, two users providing feedback about the same KB entity (and/or a simultaneous data upload) could create race conditions and/or inconsistencies within the XGS forest. Considerations like locking must receive ample attention in order to run XGS-based integration in real-time within deployed systems.

6.1.7 XGS for General Clustering

XGS is a powerful ER algorithm, and as such, it may also serve as a general purpose clustering algorithm. One advantage it has over GRINCH is that it may build more compressed trees. However, the price for maintaining arbitrary branching factor is significantly increased computation. In the context of OpenReview, we were able to sidestep

this drawback by designing models that led to wide and shallow trees (by providing strong canopy/blocking heuristics and also making models output the same score in many cases). While this worked well in the context of ER, these approaches may not be appropriate in clustering problems involving other data types. Thus, one way forward is to design models that are effective in general data domains that still produce similarly shaped trees. While our algorithm has higher computational cost than GRINCH, storing shorter trees in memory suggest that lower computational complexity may be possible. Perhaps cheaper methods of maintaining Invariant 1 exist. Alternatively, there may be methods of parallelizing some of XGS's subroutines, simplifying the method further, or developing faster mini-batch variants. Lowering the computational complexity of XGS, and its wall-clock running time, are necessary in order for it to be more readily used by practitioners.

Finally, more investigation of the benefits of XGS's shallow trees is necessary. In our work, they aided in the challenge of resolving ambiguity in the sources of KB errors. We note that shallow and wide trees also facilitate visual data exploration, which we do not touch on here. Importantly, shallow and wide trees have a small memory footprint, which is important for democratizing our method, i.e., making it accessible to practitioners without access to significant computational power. In particular, we believe that democratization of all methods—not only XGS—is an important endeavor worth significant exploration in the immediate future.

APPENDIX A

GRINCH

A.1 Proof of Theorem 1

Define the following properties:

Definition 5 (Strong Connectivity)**.** *Let $G = (\mathcal{X}, E)$ be a graph and let $\mathcal{T}[v]$ be a tree rooted at a node v with leaves, $\texttt{lvs}(v) = \mathcal{X}' \subseteq \mathcal{X}$. v is **connected** if $\mathcal{X}'$ is a connected subgraph of G. v is **strongly connected** if v and every descendant of v is connected. v is a **maximal** strongly connected node if v is strongly connected and $\texttt{par}(v)$ is not strongly connected. Finally, the tree $\mathcal{T}$ satisfies **strong connectivity** if all connected nodes in $\mathcal{T}$ are strongly connected.*

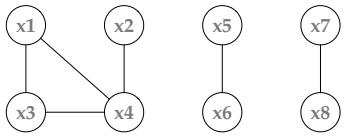
Definition 6 (Completeness)**.** *Let $G = (\mathcal{X}, E)$ be a graph and let $\mathcal{T}[v]$ be a tree rooted at a node v with leaves, $\texttt{lvs}(v) = \mathcal{X}' \subseteq \mathcal{X}$. Then v is **complete** if $\mathcal{X}'$ is a connected component in G. The tree $\mathcal{T}$ satisfies **completeness** if the set of connected components of G are (the leaves of) a tree consistent partition of $\mathcal{T}$.*

According to Theorem 1, GRINCH always constructs a tree that satisfies completeness. To prove the theorem, we will show that after the addition of each new point, the resulting tree satisfies strong connectivity and completeness. We analyze various subroutines of GRINCH and demonstrate how they preserve strong connectivity, completeness or both. In the proceeding lemmas and proofs, let $G = (\mathcal{X}, E)$ be a graph and let f be a model that separates G.

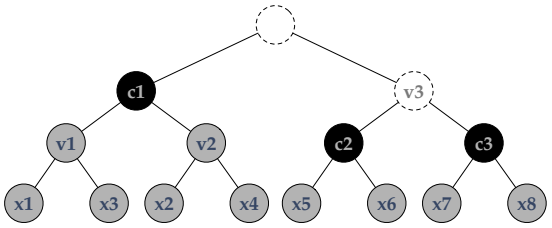
Lemma 1 (Rotation Lemma)**.** *Let $\mathcal{T}$ be a tree with $\texttt{lvs}(\mathcal{T}) = \mathcal{X}$, and let x be a new point to be added to $\mathcal{T}$. Then all nodes that were strongly connected before the addition of x are strongly connected after the addition of x, i.e., rotations preserve strong connectivity.*

Note: while rotations preserve strong connectivity, they do not guarantee completeness. Therefore rotations are insufficient for proving Theorem 1.

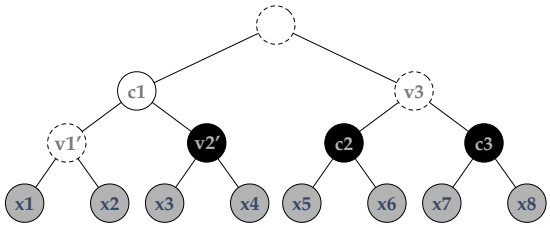
Proof. Let v be a maximal strongly connected node in $\mathcal{T}$ and assume that x is added as a leaf of v (rotations have not yet been applied). Consider two cases: (1) there exists an edge between x and some leaf in $\texttt{lvs}(v)$, and (2) there does not exist an edge between x and any leaf in $\texttt{lvs}(v)$.
Case 1: Let $L \subseteq \texttt{lvs}(v)$ be the set of v's descendant leaves to which x is connected. Then x is initially added as a sibling of its nearest neighbor leaf, x', and $x' \in L$ because f separates G. $\texttt{par}(x)$ is strongly connected because there exists an edge between x and x'.



(a) A graph $G = (\mathcal{X}, E)$.



(b) Strongly connected & complete.



(c) Complete only.

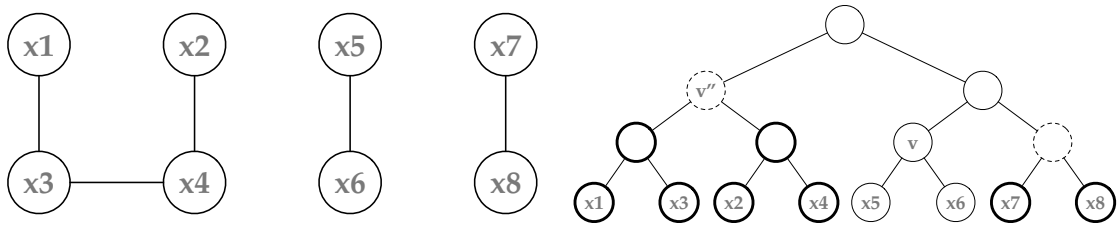
Figure A.1: A graph G with 3 connected components (Figure A.1a). In Figure A.1b and Figure A.1c, black-bordered nodes are strongly connected, thick-black-border nodes are maximal, gray bordered nodes are connected (but not strongly) and nodes with dashed borders are disconnected. The tree in Figure A.1b satisfies strong connectivity and completeness. The tree in Figure A.1c does not satisfy strong connectivity because v_1 is disconnected.

The addition of x does not disconnect v or any strongly connected descendant of v. To see why, consider the siblings of the ancestors of x' before the addition of x. Any such sibling that was connected to x', is, after the addition of x, also connected to $\texttt{par}(x)$ and thus remains strongly connected. Nodes that are not ancestors of x cannot be disconnected and thus, before rotations, strong connectivity is preserved.

Now consider subsequent rotations. By the logic above, x and its sibling, $x' = \texttt{sib}(x)$, are connected. If a rotation succeeds then x and $\texttt{aunt}(x)$ are swapped. So long as $\texttt{aunt}(x)$ and $\texttt{sib}(x)$ form a connected subgraph in G, i.e., $\phi(\texttt{sib}(x), \texttt{aunt}(x)) = \phi(x, \texttt{sib}(x)) = 1$, then the rotation preserves strong connectivity.

The only way for a rotation to disrupt strong connectivity is if x and $\texttt{aunt}(x)$ are swapped, and $\texttt{sib}(x)$ and $\texttt{aunt}(x)$ do not form a connected subgraph in G, i.e., $\phi(x, \texttt{sib}(x)) > \phi(\texttt{sib}(x), \texttt{aunt}(x))$. But, because f separates G, $\phi(x, \texttt{sib}(x)) > \phi(\texttt{sib}(x), \texttt{aunt}(x)) \implies f(x, \texttt{sib}(x)) > f(\texttt{sib}(x), \texttt{aunt}(x))$ and so, in this case, a rotation will not be performed and the procedure terminates.

Case 2: If there does not exist an edge between x and any leaf in $\texttt{lvs}(v)$, then after x is made a sibling of some leaf $x'' \in \texttt{lvs}(v)$, v is no longer strongly connected and so strong connectivity has not been preserved. Since v was strongly connected before the addition



(a) A graph $G = (\mathcal{X}, E)$.

(b) A tree on $\mathcal{X}$.

Figure A.2: v is strongly connected and maximal. An attempt to make any gray-filled node a sibling of v fails because the gray-filled nodes are strongly connected, but not maximal. An attempt to make a dashed node a sibling of v may succeed because the dashed nodes are also maximal strongly connected nodes. Merging either of the dotted nodes with v preserves strong connectivity and completeness.

of x, there exists an edge between $\texttt{lvs}(\texttt{sib}(x))$ and $\texttt{lvs}(\texttt{aunt}(x))$. Since f separates G, $f(x, \texttt{sib}(x)) < f(\texttt{sib}(x), \texttt{aunt}(x))$, which triggers the `rotate` subroutine. Rotations proceed with respect to x at least until x is no longer a descendant of v, and thus, v remains strongly connected. Strongly connected nodes that are not descendants of v are unaffected by the rotations and so strong connectivity is preserved. □

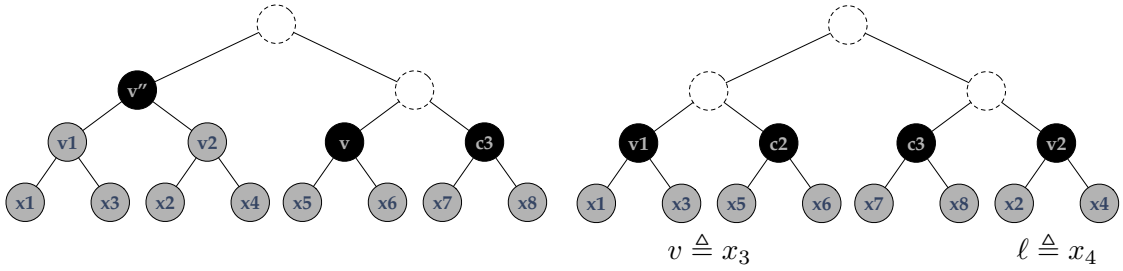
Lemma 2 (Grafting Lemma 1)**.** *Let $\mathcal{T}$ satisfy strong connectivity and completeness. Let v be a node in $\mathcal{T}$ such that v is either a maximal strongly connected node or not strongly connected. Then a* `graft` *operation initiated from v preserves strong connectivity and completeness.*

Proof. Let $\mathcal{T}$ be strongly connected and complete. Since $\texttt{lvs}(v)$ is not a strict subset of any connected component in G, there does not exist a non-empty subset s in $\texttt{lvs}(\mathcal{T})\backslash\texttt{lvs}(v)$ such that $s \cup \texttt{lvs}(v)$ is a connected subgraph in G. For any node v' that is strongly connected but not maximal, there must be an edge connecting $\texttt{lvs}(v')$ and $\texttt{lvs}(\texttt{sib}(v'))$ and $\texttt{sib}(v')$ must be strongly connected, so $f(v', \texttt{sib}(v')) > f(v, v')$. Therefore, an attempt to make any such v' the sibling of v fails.

If v'' is a maximal strongly connected node, an attempt to make v'' the sibling of v may succeed but this does not disconnect any strongly connected subtrees in $\mathcal{T}$. The same is true if v'' is not strongly connected. □

Lemma 3 (Grafting Lemma 2)**.** *Let $\mathcal{T}$ be a tree such that $\texttt{lvs}(\mathcal{T}) = \mathcal{X}$ and let $\mathcal{T}$ satisfy strong connectivity. Let v be strongly connected and let $\texttt{lvs}(v)$ be a strict subset of the vertices in some connected component, C, in G. Then, a* `graft` *initiated from v returns a node v' such that v' is strongly connected and $\texttt{lvs}(v) \subset \texttt{lvs}(v')$.*

Proof. Since $\texttt{lvs}(v)$ are a strict subset of the vertices in the connected component, C, there exists a non-empty subset s in $\texttt{lvs}(\mathcal{T})\backslash\texttt{lvs}(v)$ such that $s \cup \texttt{lvs}(v)$ constitute the vertices in C. Let ℓ maximize $f(v, \ell)$ over all $\texttt{lvs}(\mathcal{T}) \setminus \texttt{lvs}(v)$. By the fact that $\texttt{lvs}(v)$ is a strict



(a) Grafting Lemma 1 Visual Aid.

(b) Grafting Lemma 2 Visual Aid.

Figure A.3: We reuse the graph in Figure A.1a. The tree in Figure A.3a is strongly connected and complete. Consider the node v. A graft initiated from v may make v'' a sibling of v because $\texttt{lvs}(v)$ is not a (strict) subset of a connected component and v'' is maximal. After such a graft, notice that the tree would still satisfy strong connectivity and completeness. The tree in Figure A.3b is strongly connected but not complete. Consider x_3 which plays the role of v in the proof of Grafting Lemma 2. When a constrained nearest neighbor search is executed from its parent, v_1, the leaf x_4–which plays the role of ℓ–is returned. If v_1 and v_2 are made siblings, their parent is strongly connected.

subset of a connected component, there must exist an edge between $\texttt{lvs}(v)$ and ℓ. Note that ℓ is the leaf found when the constrained nearest neighbor search from v is initiated in the first line of `graft` (Algorithm 1).

If $f(v, \ell) < f(\ell, \texttt{sib}(\ell))$, then there must exist an edge between ℓ and a node in $\texttt{lvs}(\texttt{sib}(\ell))$ and so $\texttt{par}(\ell)$ is strongly connected. If $f(v, \ell) < f(v, \texttt{sib}(v))$, then there must exist an edge between a node in $\texttt{lvs}(v)$ and a node in $\texttt{lvs}(\texttt{sib}(v))$ and so $\texttt{par}(v)$ is strongly connected. In both of these cases, we do not merge v with ℓ, but instead attempt another merge between two strongly connected nodes, either: $\texttt{par}(v)$ with ℓ, v with $\texttt{par}(\ell)$, or $\texttt{par}(v)$ with $\texttt{par}(\ell)$. As before, the two nodes we are attempting to merge also have an edge between them.

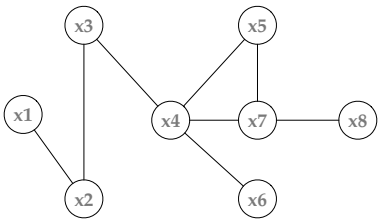
Let v_1 and v_2 be two nodes involved in a merge and let $v_1 \in \texttt{ancs}(v)$ and $v_2 \in \texttt{ancs}(\ell)$. If at some point

$$f(v_1, v_2) > \max[f(v_1, \texttt{sib}(v_1)), f(v_2, \texttt{sib}(v_2))]$$

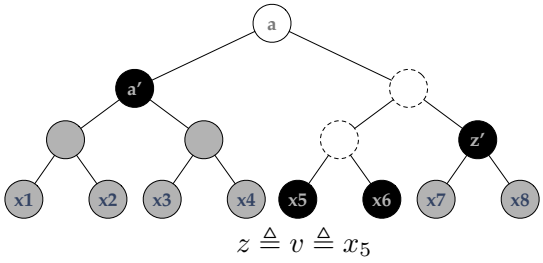
then v_2 is made a sibling of v_1 and the new parent of v_1 is returned. Since v_1 and v_2 are strongly connected and there exists an edge between $\texttt{lvs}(v_1)$ and $\texttt{lvs}(v_2)$, $\texttt{par}(v_1)$, which is created by the merge, is strongly connected, and the lemma holds.

If a merge is never performed, the recursion stops when $v_1 = v_2 = \texttt{lca}(v, \ell)$. In this case, the `lca`, which we return, is already strongly connected and, by definition, its leaves are a superset of $\texttt{lvs}(v)$. □

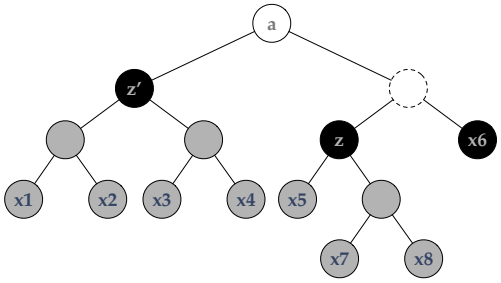
Lemma 4 (Restructuring Lemma)**.** *Let $v \in \mathcal{T}$ be strongly connected. Let $a \in \texttt{ancs}(v)$ be the deepest connected ancestor of v such that: a is not strongly connected, and all siblings of the nodes on the path from v to a are strongly connected. Then `restruct` on inputs v and a restructures $\mathcal{T}[a]$ so that a satisfies strong connectivity.*



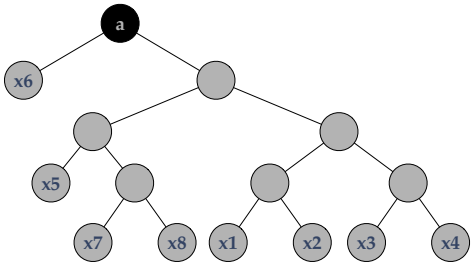
(a) A graph $G = (\mathcal{X}, E)$.



(b) a is connected, but not strongly.



(c) 1 swap applied.



(d) a is strongly connected.

Figure A.4: The `restruct` method. As before, black-filled nodes are maximal, gray-filled nodes are strongly connected, nodes with no fill and solid borders are connected (but not strongly) and nodes with dashed borders are disconnected. Also, note that the labels z, z', ℓ and a' do not apply to the same nodes throughout all figures so to match their usage in proof. In Figure A.4b, $z \triangleq v \triangleq x_5$. $\texttt{sib}(z)$ and z' are swapped to produce the tree in Figure A.4c. Finally, $\texttt{sib}(z)$ and z' from Figure A.4c are swapped to produce the tree in Figure A.4d, which is strongly connected.

Proof. Let z be the deepest ancestor of v that is strongly connected with parent $\texttt{par}(z)$ that is disconnected. Since $\texttt{par}(z)$ is disconnected (but by assumption both z and $\texttt{sib}(z)$ are connected), there are no edges between $\texttt{lvs}(z)$ and $\texttt{lvs}(\texttt{sib}(z))$.

Let a' be a child of a and without loss of generality, $a' \notin \texttt{ancs}(z)$. Since a is the deepest connected ancestor of z, there must exist an edge between $\texttt{lvs}(z)$ and $\texttt{lvs}(a')$.

When computing the argmax of $f(z, \cdot)$ in the `restruct` method, a node, z', that is connected to z will be returned and then swapped with $\texttt{sib}(z)$. The new parent of z is strongly connected because z and z' are both strongly connected and there exists an edge between $\texttt{lvs}(z)$ and $\texttt{lvs}(z')$. Any subsequent swap attempted from a disconnected node with a connected ancestor succeeds and produces a new parent that is strongly connected.

Since a is connected and a swap among the descendants of a do not change $\texttt{lvs}(a)$, swapping preserves the connectedness of a. Therefore, swaps proceed until the node a is reached at which point a must be strongly connected.

Note that a swap attempt between a strongly connected node and a node to which it is not connected fails, because f separates G. A swap attempt between a connected node and a node to which it is connected succeeds and produces a new parent that is strongly connected. □

We now prove Theorem 1.

Proof. We show by induction that if GRINCH is used to build a tree, $\mathcal{T}$, over vertices, $\mathcal{X}$, then the connected components of G are a tree consistent partition in $\mathcal{T}$. Furthermore, $\mathcal{T}$ satisfies strong connectivity.

Clearly, the theorem holds for the base case: a tree with a single node.

Let $\mathcal{X} = \texttt{lvs}(\mathcal{T})$. Assume the inductive hypothesis: that $\mathcal{T}$ satisfies completeness and strong connectivity. Now vertex x arrives.

If there does not exist an edge between x and any other vertex in $\mathcal{X}$, then after rotations, $\mathcal{T}'$ satisfies completeness. Since $\forall a \in \texttt{ancs}(x)$, $\texttt{lvs}(a)$ is a not a strict subset of any connected component in G, by Grafting Lemma 1, subsequent `graft` attempts from the ancestors of x preserve strong connectivity and completeness and so the theorem holds.

Assume that x is connected to some set of leaves $s \subseteq \texttt{lvs}(\mathcal{T})$. Since $\mathcal{T}$ satisfies strong connectivity, by the Rotation Lemma, after x is added and rotations terminate, $\mathcal{T}'$ satisfies strong connectivity. Note that $\mathcal{T}'$ may not satisfy completeness if, before the arrival of x, the leaves in s formed at least 2 distinct connected subgraphs in G.

After rotations, a series of `graft` attempts are performed. Consider the first `graft` initiated at $\texttt{par}(x)$. By Grafting Lemma 2, the attempt returns a strongly connected ancestor of x whose leaves are a strict superset of $\texttt{lvs}(x)$. If a `merge` is performed that moves a node v and makes it a sibling of v', then strong connectivity may be violated. However, notice that the only nodes that can be disconnected by such a merge are the node that, prior to the merge, were ancestors of v and also descendants of $a = \texttt{lca}(v, v')$.

After the merge, a is restructured, and by the Restructuring Lemma, the resulting tree satisfies strong connectivity. Subsequent calls to `graft` proceed from a. Notice that each invocation of `graft` returns a new strongly connected node with a strictly larger number of descendant leaves, until the resulting tree satisfies completeness. Therefore, successive grafting followed by restructuring eventually returns a node whose leaves are a connected component of G. Ultimately, after rotations and grafting, $\mathcal{T}'$ must satisfy completeness and strong connectivity. □

BIBLIOGRAPHY

[1] Dblp disambiguation page for: Wei wang. https://dblp.uni-trier.de/pers/hd/w/Wang:Wei. Accessed: 2018-05-17.

[2] A. Yosef, M., Bauer, S., Hoffart, J., Spaniol, M., and Weikum, G. HYENA: Hierarchical type classification for entity names.

[3] Angeli, G., Gupta, S., Jose, M., Manning, C. D, Ré, C., Tibshirani, J., Wu, J. Y, Wu, S., and Zhang, C. Stanford's 2014 slot filling systems. *TAC KBP* (2014).

[4] Angeli, G., Tibshirani, J., Wu, J., and Manning, C. D. Combining distant and partial supervision for relation extraction. *EMNLP* (2014).

[5] Anzaroot, S., Passos, A., Belanger, D., and McCallum, A. Learning soft linear constraints with application to citation field extraction, 2014.

[6] Backes, T. Effective unsupervised author disambiguation with relative frequencies. In *ACM/IEEE on Joint Conference on Digital Libraries* (2018), ACM.

[7] Bagga, A., and Baldwin, B. Entity-based cross-document coreferencing using the vector space model. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1* (Montreal, Quebec, Canada, Aug. 1998), Association for Computational Linguistics, pp. 79–85.

[8] Balcan, M.-F., Blum, A., and Vempala, S. A discriminative framework for clustering via similarity functions. In *Symposium on Theory of computing* (2008).

[9] Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. Open information extraction from the web.

[10] Bateni, M., Behnezhad, S., Derakhshan, M., Hajiaghayi, M., Kiveris, R., Lattanzi, S., and Mirrokni, V. Affinity clustering: Hierarchical clustering at scale. In *Advances in Neural Information Processing Systems* (2017).

[11] Bengtson, E., and Roth, D. Understanding the value of features for coreference resolution. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing* (2008), pp. 294–303.

[12] Blundell, C., and Teh, Y. W. Bayesian hierarchical community discovery. In *Advances in Neural Information Processing Systems* (2013).

[13] Blundell, C., Teh, Y. W., and Heller, K. A. Discovering non-binary hierarchical structures with bayesian rose trees. *Mixture Estimation and Applications. John Wiley & Sons* (2011).

[14] Brin, S. Extracting patterns and relations from the world wide web. In *International Workshop on The World Wide Web and Databases* (1998), Springer, pp. 172–183.

[15] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr, E. R., and Mitchell, T. M. Toward an architecture for never-ending language learning. In *AAAI* (2010).

[16] Chang, Kai-Wei, Samdani, Rajhans, and Roth, Dan. A constrained latent variable model for coreference resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (2013), pp. 601–612.

[17] Chen, D., Fisch, A., Weston, J., and Bordes, A. Reading wikipedia to answer open-domain questions. In *ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)* (Jan. 2017), Association for Computational Linguistics (ACL), pp. 1870–1879.

[18] Clark, K., and Manning, C. D. Deep reinforcement learning for Mention-Ranking coreference models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (Nov. 2016), pp. 2256–2262.

[19] Clark, K., and Manning, C. D. Improving coreference resolution by learning entity-level distributed representations. In *Association for Computational Linguistics* (2016).

[20] Cohen-Addad, V., Kanade, V., Mallmann-Trenn, F., and Mathieu, C. Hierarchical clustering: Objective functions and algorithms. In *Symposium on Discrete Algorithms* (2018).

[21] Culotta, A., Kanani, P., Hall, R., Wick, M., and McCallum, A. Author disambiguation using error-driven machine learning with a ranking loss function. In *Workshop on Information Integration on the Web* (2007).

[22] Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., and McCallum, A. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations* (2018).

[23] Das, Rajarshi, Dhuliawala, Shehzaad, Zaheer, Manzil, and McCallum, Andrew. Multi-step retriever-reader interaction for scalable open-domain question answering. In *International Conference on Learning Representations* (2019).

[24] Dasgupta, S. A cost function for similarity-based hierarchical clustering. *arXiv:1510.05043* (2015).

[25] Del Corro, L., Abujabal, A., Gemulla, R., and Weikum, G. FINET: Context-Aware Fine-Grained named entity typing, 2015.

[26] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. *KDD* (2014).

[27] Dong, X. L., Gabrilovich, E., Heitz, G., Horn, W., Murphy, K., Sun, S., and Zhang, W. From data fusion to knowledge fusion. *VLDB* (2014).

[28] Durrett, G., Hall, D., and Klein, D. Decentralized entity-level modeling for coreference resolution. In *Association for Computational Linguistics* (2013).

[29] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (1996).

[30] Etzioni, O., Banko, M., Soderland, S., and Weld, D. S. Open information extraction from the web. *Communications of the ACM 51*, 12 (2008), 68–74.

[31] Fichtenberger, H., Gillé, M., Schmidt, M., Schwiegelshohn, C., and Sohler, C. Bico: Birch meets coresets for k-means clustering. In *European Symposium on Algorithms* (2013).

[32] Firmani, D., Saha, B., and Srivastava, D. Online entity resolution using an oracle. *VLDB* (2016).

[33] Galárraga, L., Heitz, G., Murphy, K., and Suchanek, F. M. Canonicalizing open knowledge bases. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management* (New York, NY, USA, Nov. 2014), CIKM '14, Association for Computing Machinery, pp. 1679–1688.

[34] Ganchev, K., Graça, J., Gillenwater, J., and Taskar, B. Posterior regularization for structured latent variable models. *The Journal of Machine* (2010).

[35] Geusebroek, J., Burghouts, G. J., and Smeulders, A. W.M. The amsterdam library of object images. *International Journal of Computer Vision* (2005).

[36] Gooi, C. H., and Allan, J. Cross-document coreference on a large scale corpus. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004* (2004), pp. 9–16.

[37] Greenberg, C. S., Bansé, D., Doddington, G. R., Garcia-Romero, D., Godfrey, J. J., Kinnunen, T., Martin, A. F., McCree, A., Przybocki, M., and Reynolds, D. A. The nist 2014 speaker recognition i-vector machine learning challenge. In *Odyssey: The Speaker and Language Recognition Workshop* (2014).

[38] Grishman, R., and Sundheim, B. Message understanding conference-6: a brief history. In *Proceedings of the 16th conference on Computational linguistics - Volume 1* (USA, Aug. 1996), COLING '96, Association for Computational Linguistics, pp. 466–471.

[39] Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M.-W. REALM: Retrieval-Augmented language model Pre-Training.

[40] Haghighi, A., and Klein, D. Unsupervised coreference resolution in a nonparametric bayesian model. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics* (2007), pp. 848–855.

[41] Haghighi, A., and Klein, D. Coreference resolution in a modular, entity-centered model. In *Human Language Technologies: Association for Computational Linguistics* (2010).

[42] Han, H., Zha, H., and Giles, C. L. Name disambiguation in author citations using a k-way spectral clustering method. In *Joint Conference on Digital Libraries* (2005).

[43] Hasegawa, T., Sekine, S., and Grishman, R. Discovering relations among named entities from large corpora. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)* (2004), pp. 415–422.

[44] Heller, K., and Ghahramani, Z. Randomized algorithms for fast bayesian hierarchical clustering.

[45] Heller, K. A., and Ghahramani, Z. Bayesian hierarchical clustering. In *International conference on Machine Learning* (2005).

[46] Hoffart, J., Suchanek, F. M, Berberich, K., and Weikum, G. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence* (2013).

[47] Hu, X., Rudin, C., and Seltzer, M.o. Optimal sparse decision trees.

[48] Huang, J., Ertekin, S., and Giles, C. L. Efficient name disambiguation for large-scale databases. In *European conference on principles of data mining and knowledge discovery* (2006).

[49] Jain, S., and Neal, R. M. A split-merge markov chain monte carlo procedure for the dirichlet process mixture model. *Journal of computational and Graphical Statistics* (2004).

[50] Kambhatla, N. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions* (2004), pp. 22–es.

[51] Khabsa, M., Treeratpituk, P., and Giles, C. L. Large scale author name disambiguation in digital libraries. In *International Conference on Big Data (Big Data)* (2014).

[52] Kim, K., Khabsa, M., and Giles, C. L. Inventor name disambiguation for a patent database using a random forest and dbscan. In *Joint Conference on Digital Libraries* (2016).

[53] Kleinberg, J. The small-world phenomenon: An algorithmic perspective. In *Symposium on Theory of computing* (2000).

[54] Kleinberg, J. Complex networks and decentralized search algorithms. In *International Congress of Mathematicians (ICM)* (2006).

[55] Kobren, A., Monath, N., Krishnamurthy, A., and McCallum, A. A hierarchical algorithm for extreme clustering. In *Knowledge Discovery and Data Mining* (2017).

[56] Kobren, A., Monath, N., and McCallum, A. Entity-centric attribute feedback. In *Workshop on Automated Knowledge Base Construction* (2017).

[57] Kobren, A., Monath, N., and McCallum, A. Integrating user feedback under identity uncertainty in knowledge base construction. In *Automated Knowledge Base Construction* (2019).

[58] Kohli, Pushmeet, Torr, Philip HS, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision* (2009).

[59] Krishnamurthy, A., Balakrishnan, S., Xu, M., and Singh, A. Efficient active algorithms for hierarchical clustering. *International Conference on Machine Learning* (2012).

[60] Lakkaraju, H., Bach, S. H., and Leskovec, J. Interpretable decision sets: A joint framework for description and prediction. *KDD 2016* (Aug. 2016), 1675–1684.

[61] Lee, C., Hwang, Y.-G., Oh, H.-J., Lim, S., Heo, J., Lee, C.-H., Kim, H.-J., Wang, J.-H., and Jang, M.-G. Fine-Grained named entity recognition using conditional random fields for question answering. In *Information Retrieval Technology* (2006), Springer Berlin Heidelberg, pp. 581–587.

[62] Lee, H., Recasens, M., Chang, A., Surdeanu, M., and Jurafsky, D. Joint entity and event coreference resolution across documents. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (2012).

[63] Lee, K., He, L., Lewis, M., and Zettlemoyer, L. End-to-end neural coreference resolution.

[64] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web* (2015).

[65] Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D., and Shepherd, M. Cyc: toward programs with common sense. *Commun. ACM 33*, 8 (Aug. 1990), 30–49.

[66] Levin, M., Krawczyk, S., Bethard, S., and Jurafsky, D. Citation-based bootstrapping for large-scale author disambiguation. *American Society for Information Science and Technology* (2012).

[67] Li, F., Dong, X. L., Langen, A., and Li, Y. Knowledge verification for long-tail verticals. *VLDB* (2017).

[68] Li, G. Human-in-the-loop data integration. *VLDB* (2017).

[69] Liberty, E., Sriharsha, R., and Sviridenko, M. An algorithm for online k-means clustering. In *Workshop on Algorithm Engineering and Experiments* (2016).

[70] Ling, X., and Weld, D. S. Fine-grained entity recognition. *Twenty-Sixth AAAI Conference on Artificial Intelligence* (2012).

[71] Liu, W., Islamaj Doğan, R., Kim, S., Comeau, D. C., Kim, W., Yeganova, L., Lu, Z., and Wilbur, W. J. Author name disambiguation for pub med. *Information Science and Technology* (2014).

[72] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. RoBERTa: A robustly optimized BERT pretraining approach.

[73] Malkov, Y., Ponomarenko, A., Logvinov, A., and Krylov, V. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* (2014).

[74] Mann, G. S., and McCallum, A. Generalized expectation criteria for semi-supervised learning of conditional random fields. In *Proceedings of ACL-08: HLT* (Columbus, Ohio, June 2008), Association for Computational Linguistics, pp. 870–878.

[75] Mazumdar, A., and Saha, B. Clustering via crowdsourcing. *arXiv:1604.01839* (2016).

[76] Mazumdar, A., and Saha, B. A theoretical analysis of first heuristics of crowdsourced entity resolution. In *AAAI Conference on Artificial Intelligence* (2017).

[77] McCallum, A., and Wellner, B. Conditional models of identity uncertainty with application to noun coreference. In *Advances in neural information processing systems* (2005).

[78] Mintz, M., Bills, S., Snow, R., and Jurafsky, D. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP* (2009), pp. 1003–1011.

[79] Monath, N., Kobren, A., Krishnamurthy, A., Glass, M., and McCallum, A. Scalable hierarchical clustering with tree grafting. In *International Conference on Knowledge Discovery & Data Mining* (2019).

[80] Murty, S., Verga, P., Vilnis, L., Radovanovic, I., and McCallum, A. Hierarchical losses and new resources for fine-grained entity typing and linking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2018), pp. 97–109.

[81] Nadeau, D., and Sekine, S. A survey of named entity recognition and classification. *Lingvisticæ Investigationes 30*, 1 (Jan. 2007), 3–26.

[82] Ng, V. Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the 48th annual meeting of the association for computational linguistics* (2010), pp. 1396–1411.

[83] Ng, V., and Cardie, C. Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* (2002), pp. 104–111.

[84] Pasula, H., Marthi, B., Milch, B., Russell, S. J., and Shpitser, I. Identity uncertainty and citation matching. In *Advances in neural information processing systems* (2003).

[85] Pawar, S., Palshikar, G. K., and Bhattacharyya, P. Relation extraction : A survey.

[86] Pedersen, T., Kulkarni, A., Angheluta, R., Kozareva, Z., and Solorio, T. An unsupervised language independent method of name discrimination using second order co-occurrence features. In *Computational Linguistics and Intelligent Text Processing - 7th International Conference, CICLing 2006, Proceedings* (July 2006), pp. 208–222.

[87] Petroni, Fabio, Lewis, Patrick, Piktus, Aleksandra, Rocktäschel, Tim, Wu, Yuxiang, Miller, Alexander H., and Riedel, Sebastian. How context affects language models' factual predictions. In *Automated Knowledge Base Construction* (2020).

[88] Raghunathan, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D., and Manning, C. A multi-pass sieve for coreference resolution. In *Empirical Methods in Natural Language Processing* (2010).

[89] Rao, D., McNamee, P., and Dredze, M. Streaming cross document entity coreference resolution. In *International Conference on Computational Linguistics: Posters* (2010).

[90] Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. Snorkel: Rapid training data creation with weak supervision. *Proceedings VLDB Endowment 11*, 3 (Nov. 2017), 269–282.

[91] Ravin, Y., and Kazi, Z. Is hillary rodham clinton the president? disambiguating names across documents. In *Coreference and Its Applications* (1999).

[92] Ré, C., Sadeghian, A. A., Shan, Z., Shin, J., Wang, F., Wu, S., and Zhang, C. Feature engineering for knowledge base construction. *arXiv:1407.6439* (2014).

[93] Riedel, S., Yao, L., McCallum, A., and Marlin, B. M. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2013), pp. 74–84.

[94] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* (2015).

[95] Sarawagi, S. *Information extraction*. Now Publishers Inc, 2008.

[96] Sevgili, O., Shelmanov, A., Arkhipov, M., Panchenko, A., and Biemann, C. Neural entity linking: A survey of models based on deep learning.

[97] Shen, W., Wang, J., and Han, J. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Trans. Knowl. Data Eng. 27*, 2 (Feb. 2015), 443–460.

[98] Shimaoka, S., Stenetorp, P., Inui, K., and Riedel, S. Neural architectures for fine-grained entity type classification.

[99] Singh, S., Subramanya, A., Pereira, F., and McCallum, A. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Association for Computational Linguistics: Human Language Technologies* (2011).

[100] Singh, Sameer, Wick, Michael L., and McCallum, Andrew. Distantly labeling data for large scale cross-document coreference. *ArXiv abs/1005.4298* (2010).

[101] Steorts, R. C., Hall, R., and Fienberg, S. E. A bayesian approach to graphical record linkage and deduplication. *Journal of the American Statistical Association* (2016).

[102] Thai, D., Xu, Z., Monath, N., Veytsman, B., and McCallum, A. Using bibtex to automatically generate labeled data for citation field extraction. In *Automated Knowledge Base Construction* (2020).

[103] Torvik, V. I., and Smalheiser, N. R. Author name disambiguation in medline. *ACM Transactions on Knowledge Discovery from Data* (2009).

[104] Torvik, V. I., Weeber, M., Swanson, D. R., and Smalheiser, N. R. A probabilistic similarity metric for medline records: A model for author name disambiguation. *Journal of the American Society for information science and technology* (2005).

[105] Treeratpituk, P., and Giles, C. L. Disambiguating authors in academic publications using random forests. In *Joint conference on Digital libraries* (2009).

[106] Ustun, B., Tracà, S., and Rudin, C. Supersparse linear integer models for interpretable classification.

[107] Vashishth, S., Jain, P., and Talukdar, P. CESI: Canonicalizing open knowledge bases using embeddings and side information. In *Proceedings of the 2018 World Wide Web Conference* (Republic and Canton of Geneva, CHE, Apr. 2018), WWW '18, International World Wide Web Conferences Steering Committee, pp. 1317–1327.

[108] Vesdapunt, N., Bellare, K., and Dalvi, N. Crowdsourcing algorithms for entity resolution. *VLDB* (2014).

[109] Vrandečić, D., and Krötzsch, M. Wikidata: a free collaborative knowledge base.

[110] Walker, L. A., and Armstrong, M. " I cannot tell what the dickens his name is": Name disambiguation in institutional repositories. *Journal of Librarianship and Scholarly Communication* (2014).

[111] Wang, D., and Wang, Y. An improved cost function for hierarchical cluster trees. *CoRR* (2018).

[112] Wang, H., Wan, R., Wen, C., Li, S., Jia, Y., Zhang, W., and Wang, X. Author name disambiguation on heterogeneous information network with adversarial representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 238–245.

[113] Wang, J., Kraska, T., Franklin, M. J., and Feng, J. Crowder: Crowdsourcing entity resolution. *VLDB* (2012).

[114] Wang, J., Li, G., Kraska, T., Franklin, M. J., and Feng, J. Leveraging transitive relations for crowdsourced joins. In *International Conference on Management of Data* (2013).

[115] Wang, X., Dong, X. L., and Meliou, A. Data x-ray: A diagnostic tool for data errors. In *ICDM* (2015).

[116] Watts, D. J., and Strogatz, S. H. Collective dynamics of 'small-world'networks. *nature* (1998).

[117] Wellner, B., McCallum, A., Peng, F., and Hay, M. An integrated, conditional model of information extraction and coreference with application to citation matching.

[118] Wick, M., Kobren, A., and McCallum, A. Probabilistic reasoning about human edits in information integration. In *Machine Learning Meets Crowdsourcing* (2013).

[119] Wick, M., Schultz, K., and McCallum, A. Human-machine cooperation with epistemological dbs: supporting user corrections to knowledge bases. In *Automatic Knowledge Base Construction and Web-scale Knowledge Extraction* (2012).

[120] Wick, M., Singh, S., and McCallum, A. A discriminative hierarchical model for fast coreference at large scale. In *ACL* (2012).

[121] Wick, Michael Louis. *Epistemological Databases for Probabilistic Knowledge Base Construction*. PhD thesis, University of Massachusetts Amherst, 2015.

[122] Wiseman, S., Rush, A. M., and Shieber, S. M. Learning global features for coreference resolution. *NAACL-HLT* (2016).

[123] Yadav, V., and Bethard, S. A survey on recent advances in named entity recognition from deep learning models.

[124] Yates, A., and Etzioni, O. Unsupervised resolution of objects and relations on the web. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference* (2007), pp. 121–130.

[125] Zhang, L., Song, M., Liu, Z., Liu, X., Bu, J., and Chen, C. Probabilistic graphlet cut: Exploiting spatial structure cue for weakly supervised image segmentation. In *Computer Vision and Pattern Recognition* (2013).

[126] Zhang, T., Ramakrishnan, R., and Livny, M. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record* (1996).

[127] Zhang, Y., Zhang, F., Yao, P., and Tang, J. Name disambiguation in aminer: Clustering, maintenance, and human in the loop. In *Knowledge Discovery & Data Mining* (2018).

[128] Zheng, S., Wang, F., Bao, H., Hao, Y., Zhou, P., and Xu, B. Joint extraction of entities and relations based on a novel tagging scheme. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Vancouver, Canada, July 2017), Association for Computational Linguistics, pp. 1227–1236.